Universidad Pública de Navarra

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS AGRONOMOS

Nafarroako Unibertsitate Publikoa

NEKAZARITZAKO INGENIARIEN

GOI MAILAKO ESKOLA TEKNIKOA

TRABAJO FIN DE MÁSTER

MEJORA E IMPLEMENTACIÓN DE NUEVAS FUNCIONALIDADES A UN VISOR EN LA EMPRESA KUNAK TECHNOLOGIES S.L.

Presentado por:

Iban Iturria Aguinaga

Máster Universitario en SIG y Teledetección

1. ABSTRACT

The new technologies have made web mapping an indispensable tool in the value enhancement of data. In addition, it has proven to be a tool capable of simulating work environments similar to desktop GIS. Here we show the case of the web viewer of the company Kunak Technologíes, with the aim of improving the visualization of the markers, improving the representation of clusters and adding new functionalities such as, interpolation and search tools, among others. All these modifications Will be developed using the library for web mapping leaflet. The results obtained have been very satisfactory: the markers change in size depending on the zoom level and showing relevant data; the clusters provide more information about the markers; and the new functionalities fulfill their function. We conclude that both the improvements and the new functionalities have provided more clarity, dynamism and usefulness to the viewer.

2. RESUMEN

Las nuevas tecnologías han hecho del web mapping una herramienta indispensable en la puesta en valor de los datos. Además, ha demostrado ser una herramienta capaz de simular entornos de trabajo similares a los GIS de escritorio. Aquí se muestra el caso del visor web de la empresa Kunak Technologíes, con el objetivo de mejorar la visualización de los marcadores, mejorar la representación de clústeres y adición de nuevas funcionalidades como son la interpolación y herramientas de búsqueda, entre otros. Todos estos cambios se realizarán utilizando la librería para web mapping Leaflet. Los resultados obtenidos han sido muy satisfactorios: los marcadores cambien de tamaño en función del nivel de zoom y muestran datos relevantes; los cústeres aportan más información sobre los marcadores; y las nuevas funcionalidades cumplen con su función. Se concluye que tanto las mejoras realizadas como las funcionalidades añadidas aportan mayor claridad, dinamismo y utilidad al visor.

Fdo: Daniel Paternain Dallo

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN Y OBJETIVOS

Los avances tecnológicos que se han dado en los últimos años, permiten una recogida masiva de datos en ámbitos muy diversos. En este sentido cada vez es más importante el tratamiento y visualización de los mismos para poder obtener información y conocimiento.

Dado que muchos de estos datos cuentan con un componente espacial, es decir, son georreferenciables mediante coordenadas, el web mapping está siendo una herramienta muy frecuente para la visualización de los datos (Sack, 2018). El web mapping es el proceso de utilizar mapas servidos por sistemas de información geográfica (SIG) en una página web, que es más que simple cartografía web, permitiendo a los usuarios interactuar con el mapa de múltiples formas. Crear mapas web requiere, al menos, conocimientos básicos de programación y desarrollo web, siendo vitales en el futuro del GIS y la cartografía. Mientras que gran parte de programas de cartografía y SIG continúan centrándose en aplicaciones de escritorio orientados a pulsar botones, con muy poca aplicación para generar código basado en texto (Sack, 2018), el web mapping ofrece una alternativa funcional y más personalizable.

Las páginas web están pasando de ser simples escaparates de información a incluir herramientas y funcionalidades cada vez más complejas. Estas funcionalidades pueden ser muy diversas, pero el hecho de tener un visor permite generar herramientas y aplicaciones capaces de crear un entorno similar al de programas o aplicaciones de SIG. Tal es así que incluso organismos públicos utilizan este tipo de aplicaciones para exponer sus datos, dando transparencia a los mismos al dejarlos al alcance de los usuarios para que puedan manipularlos y reutilizarlos (Elwood & Leszczynski, 2013). Al mismo tiempo permite tanto a la comunidad (Eanes, Silbernagel, Hart, Robinson, & Axler, 2018), como a empresas (grandes y pequeñas) o entidades sin ánimo de lucro, poder generar valor en sus datos para tomar decisiones propias y/o ayudar a tomar decisiones locales (Giannaros et al., 2018; Knoth, Slimani, Appel, & Pebesma, 2018; Martin, 2004).

La reutilización de los datos, promueve la estandarización de los mismos, de tal forma que cualquier usuario sea capaz de consumirlos independientemente del medio que utilice. En este punto, el Open Geospatial Consortium (OGC), ha definido diferentes estándares referidos a formatos de imagen geoespacial, metadatos cartográficos y convenciones de mapas geológicos que se han aceptado por diferentes instituciones de investigación cartográfica (Hare, Rossi, Frigeri, & Marmo, 2018).

Esto permite tener información de cualquier parte del mundo en tiempo real, muy útil en aplicaciones donde se necesita una rápida toma de decisiones (Wang & Cheng, 2008), como las enfocadas a monitorizar el medio ambiente (Savini et al., 2018), la cobertura terrestre (Melis et al., 2018) o en apoyo a ayudas humanitarias (Knoth et al., 2018).

En este estudio se presenta el caso de la empresa Kunak Technologies S.L. (KUNAK). KUNAK es una Start Up innovadora que desarrolla equipos electrónicos y de radiofrecuencia de muy bajo consumo de energía para monitorización y control remoto, abarcando desde el diseño del hardware, hasta el firmware, protocolos, Software y servicios en la nube.

Sus productos y soluciones van enfocados a la Industria 4.0, M2M, IIoT, las infraestructuras, smart cities y medio ambiente. Su sede está ubicada en el Vivero de Innovación de CEIN y tiene clientes como Organización Mundial de la Salud-OMS, la Agencia de Protección Medioambiental de los Estados Unidos, Volkswagen, Suez, el Canal de Isabel II, Ormazabal Velatia o Acciona Energía.

Kunak Technologies, es una de las 5 empresas navarras que han logrado financiación del programa Fase II Horizonte 2020, que es el principal instrumento financiero de la Unión Europea para I+D+i entre 2014-2020. Desde la fecha de inicio solo 842 proyectos han conseguido financiación de los más de 16.000 presentados, entre las que se encuentra Kunak (NavarraCapital.es, 2018).

Entre los productos que ofrecen se dividen en Air Quality Monitoring, Dataloggers 2G/NB-IoT/LTE, Redes Radio LoRa k, M2M IoT OEM y Data Managements & Collection tal y como indican en su página web (figura 1)

Figura 1. Productos ofrecidos por Kunak en su web. (Kunak, 2018)

La mayor parte de los equipos (sensores) desplegados por Kunak están orientados a la monitorización de la calidad del aire. Los dispositivos recogen datos enviándolos a la nube y son ofrecidos a sus clientes mediante una plataforma web, donde se ubica un visor mostrando los equipos a través de marcadores.

1.1 OBJETIVOS

El objetivo de este trabajo es modificar el visualizador web de la empresa Kunak para mejorar la visualización de los dispositivos, añadiendo al mismo tiempo nuevas funcionalidades sobre la visualización de los mismos.

El objetivo de este trabajo se centra en la modificación de tres aspectos concretos del visualizador: los marcadores de los dispositivos, la visualización de clústeres de dispositivos y la funcionalidad de interpolación en las medidas de los dispositivos. Además de estas tres líneas de actuación., se contemplan otras modificaciones menores que, aun no estando previstas en los objetivos principales del trabajo, repercutan en una mejora clara de la funcionalidad y la calidad del visor web.

Así pues, el objetivo del trabajo puede dividirse en tres subobjetivos:

- **Modificaciones de los marcadores:** Modificar los marcadores actuales a unos nuevos que muestren el valor de algún parámetro medido por los sensores, manteniendo el sistema de colores que hay actualmente para identificar el estado del marcador. Con esto se pretende mejorar la visualización de los datos directamente desde el visor, para reducir el esfuerzo del usuario para consultar información.

- **Modificación de la visualización de los clústeres:** Modificar la apariencia de los clústeres de tal forma que aporten más información de los marcadores que están agrupando y faciliten o homogeneicen la representación de los mismos en el visor. Con esto se complementa el punto anterior, pudiendo no solo ver los valores de los marcadores, sino una visión instantánea del estado de los mismos por zonas, mejorando la apariencia y visualización de los dispositivos.

- **Añadir la funcionalidad de interpolación:** Se pretende realizar una interpolación de los datos obtenidos de los sensores. Este proceso se realizará seleccionando un área de interpolación del visor web, utilizando los puntos que caen dentro de dicha superficie. Con esto se pretende obtener una imagen ráster o un vectorial para añadir al visor y tener una ligera aproximación de cómo se comporta el parámetro medido en el espacio, sin necesitar una gran precisión.

2. SITUACIÓN DE PARTIDA

En esta sección se describe brevemente cómo era la situación de partida del visor web antes de la realización del presente trabajo.

2.1 RECOGIDA DE DATOS Y BASE DE DATOS

Debido a que la empresa se dedica a diversas áreas, la tipología de los datos recogidos es diferente en función del dispositivo y aplicación. Sin embargo, todos ellos se comunican con la nube dando a los usuarios los datos que recoge el dispositivo en tiempo real mediante tarjetas sim integradas en los dispositivos.

Actualmente se cuenta con una base de datos relacional MySQL con 37 relaciones (tablas), que almacenan información diferente, desde lecturas de dispositivos hasta datos de usuarios o eventos (figura 2).

Figura 2. Pantallazo de las relaciones que forman la base de datos de la empresa.

Esta base de datos no tiene añadida componente espacial, por lo que las coordenadas de las entradas no están en una columna de geometría (como lo estarían de lo contrario), sino que tienen atributos independientes para longitud y latitud.

Las tablas que se han consultado para realizar el trabajo son las de device, element ID y read.

En device se encuentran los datos de los dispositivos, cuyos atributos consultados para el visor son ID, serial_number, latitude, longitude, original_tag, is_gateway. De esta tabla se obtienen las coordenadas de los dispositivos que no tienen GPS y se introducen en el visor, añadiendo el resto de parámetros citados como atributos. Tanto el atributo ID como Serial_number son únicos siendo el ID la clave primaria de la tabla.

En element ID se encuentran los identificadores que hacen referencia al tipo de información que recoge cada dispositivo, es decir, si un dispositivo mide más de un contaminante del aire, tendrá asociados a la ID del dispositivo diferentes ID de elementos. Si los dispositivos llevan GPS las lecturas de los de las coordenadas también son almacenados en esta tabla con ID 3 y 4 respectivamente para longitud y latitud. De esta tabla se puede conocer qué mide cada dispositivo, o dicho de otra forma que dispositivos están midiendo determinado parámetro.

En la tabla read se encuentran las lecturas de los dispositivos, por lo tanto, conociendo los dispositivos de los que se quiere obtener la información y el tipo de elemento que se quiere consultar, se obtienen las lecturas del elemento en el dispositivo deseado, ya que las claves para acceder a las lecturas son la ID del dispositivo y la ID del elemento de consulta además de la fecha.

La página web es el escaparate de esta base de datos que expone al usuario la información de forma clara, permitiendo además realizar ciertas operaciones.

2.2 PÁGINA WEB

Para acceder a la página web es necesario tener credenciales, puesto que exige tener un usuario y contraseña. Tal y como se puede observar en la figura 3, en primera instancia se muestra una página con un menú en la parte superior donde se puede escoger entre, Home, Alarms, Data, Configuration, Management y Tools. A su vez en el lado izquierdo se ve una columna en la que se muestran los dispositivos desplegados con un color indicando su estado.

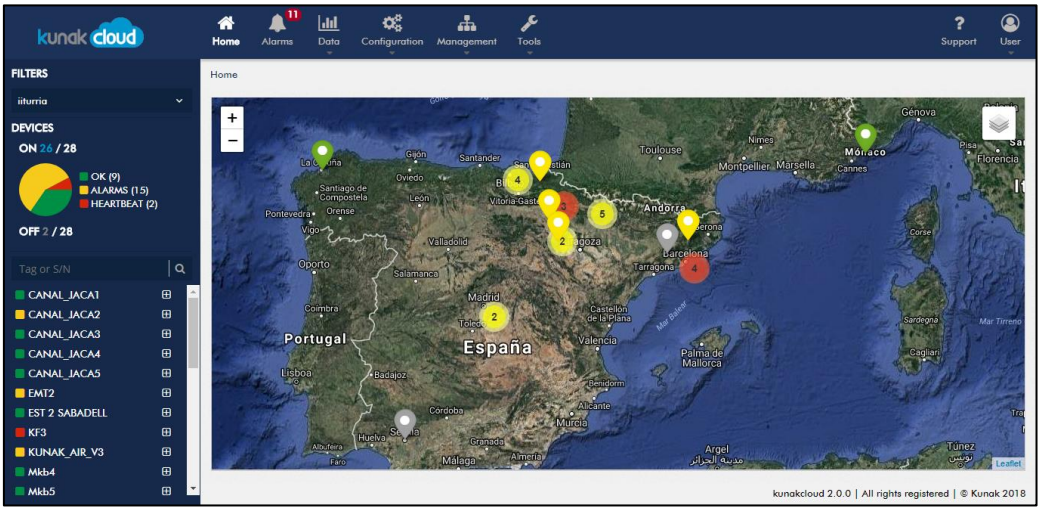


Figura 3. Página Web.

La pestaña de Alarms permite visualizar y gestionar las alarmas que se han producido en el visor. La información se muestra en una tabla en la que se especifica si ha sido o no confirmada, el dispositivo en el que se ha dado, el sensor, una breve descripción, la fecha de inicio y la fecha de fin. Además se pueden filtrar las alarmas por dispositivo, sensor y periodo (figura 4).

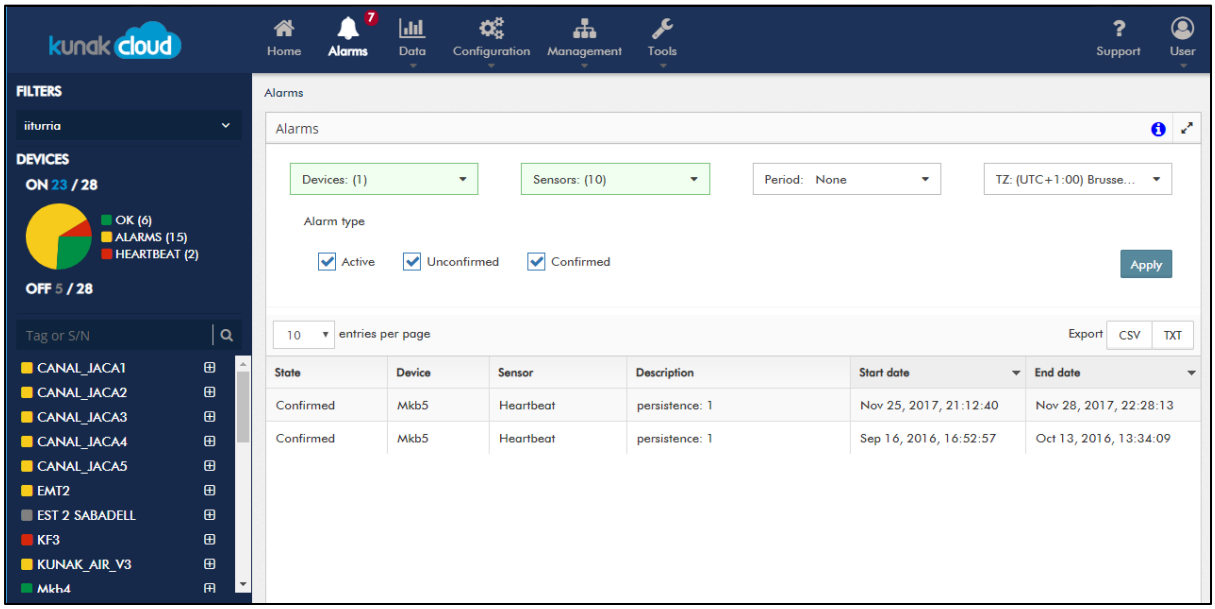


Figura 4. Pestaña de alarmas de la página web KunakCloud.

En el apartado data se muestran datos más detallados de los señores de los dispositivos. La página web permite generar gráficas como la que se muestran en la figura 5, seleccionando los dispositivos implicados en el cálculo, los parámetros a medir y el periodo.

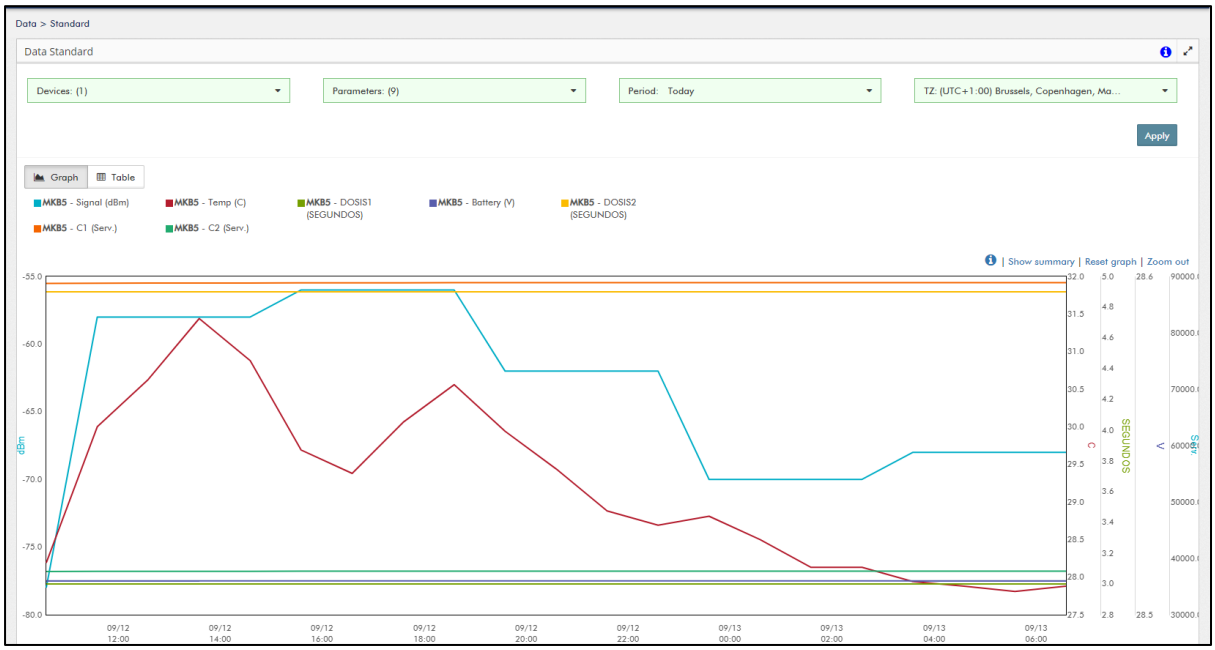


Figura 5. Generación de gráficas partiendo de los datos de los dispositivos.

Además del gráfico, también presenta un pequeño resumen que aporta más información sobre los dispositivos seleccionados (figura 6).

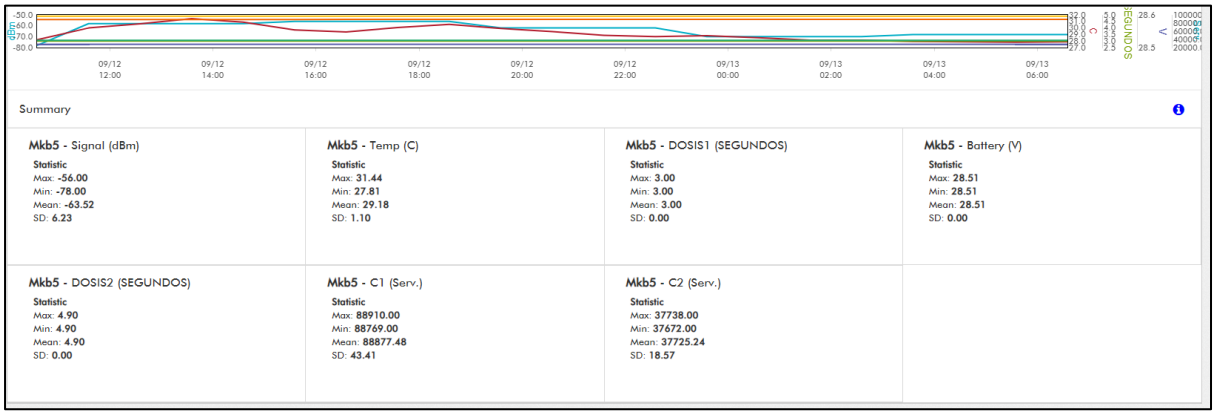


Figura 6. Resumen que genera la pestaña de datos cuando se pide a la página web que calcule las estadísticas de uno o varios dispositivos.

Con la pestaña "configuration" se permite configurar/programar los dispositivos de forma remota, esta pestaña da la opción de simple para configurar un solo dispositivo, o múltiple para configurar un grupo de ellos al mismo tiempo (figura 7).

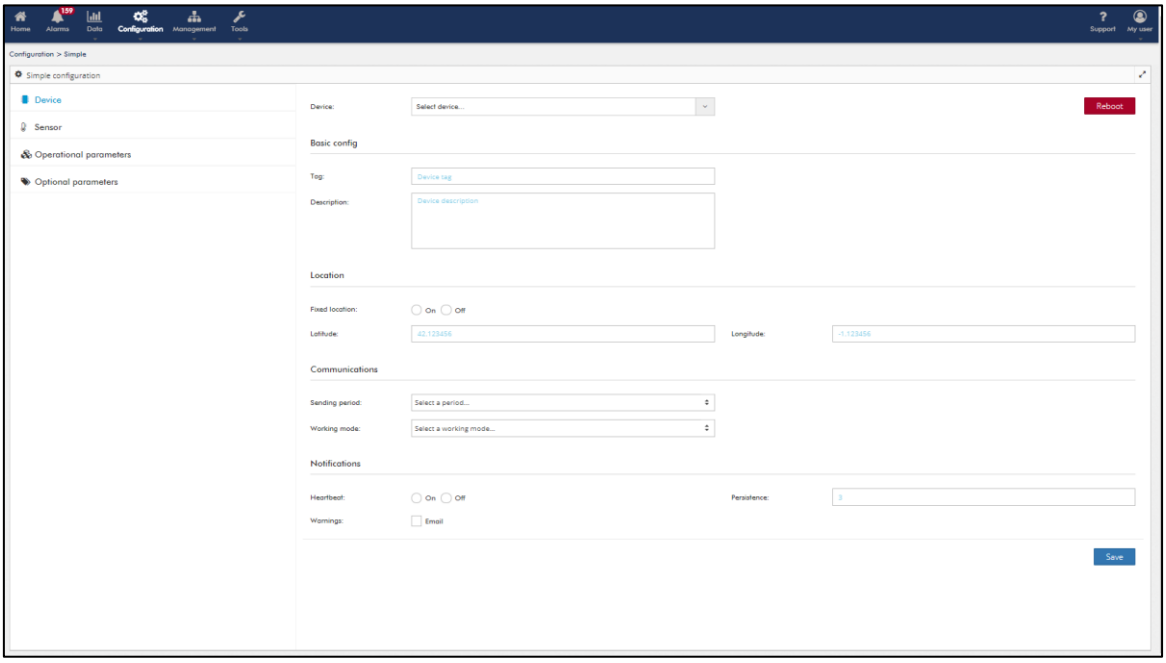


Figura 7. Pantallazo de la página de configuración de dispositivos de KunakCloud.

La pestaña de Management tiene dos posibles accesos, "Users" y "Devices". "Users" sirve para gestionar los accesos y permisos que tiene cada usuario en la aplicación. Esto puede ser utilizado tanto por la propia empresa para dar de alta a nuevos usuarios, como por un cliente que quiere gestionar los permisos que tendrán sus trabajadores. Mientras que la pestaña de "devices" sirve para dar de alta nuevos dispositivos y asignarlos a diferentes usuarios. Esto permite que cada usuario tenga acceso a ciertos dispositivos y no a todos (figura 8).

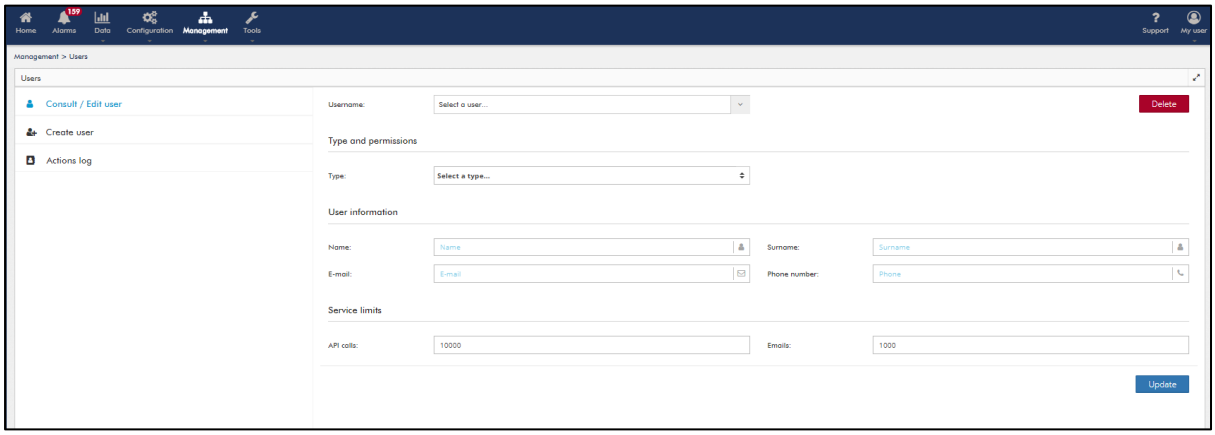


Figura 8. Pantallazo de la pestaña "Management/users" de la web de KunakCloud.

Por último, la pestaña ”Tools” permite realizar la calibración remota de los dispositivos. El algoritmo que se utiliza para la calibración es parte del conocimiento de la empresa y por eso no es publicada. Esta pestaña tiene la apariencia que se puede ver en la figura 9.

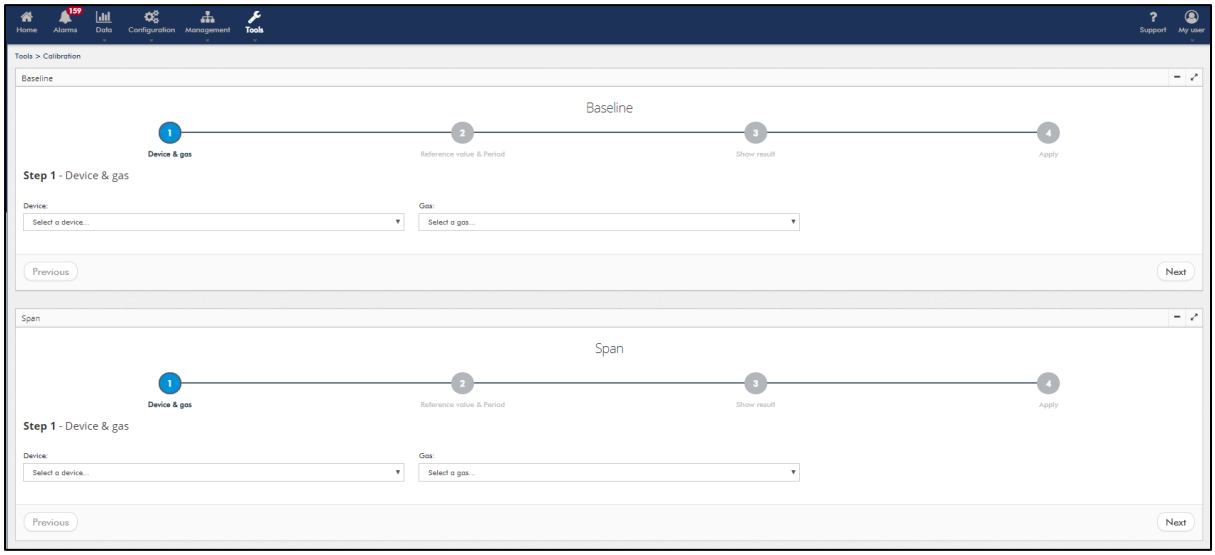


Figura 9. Pantallazo de la pestaña "Tools" de la web de KunakCloud.

2.3 VISOR WEB

En el espacio que queda entre el menú superior y la columna izquierda se encuentra el visor web, donde se muestra un mapa de fondo de satélite u open streetmap (a escoger por el usuario) y marcadores que corresponden con la ubicación de los dispositivos.

Este visor está desarrollado en JavaScript utilizando la librería leaflet para web mapping. Actualmente el visor cuenta con unos marcadores simples que únicamente muestran la ubicación de los equipos en un mapa con el sistema de coordenadas WGS84. Estos marcadores toman diferentes colores en función del estado del equipo siendo rojo si el equipo no se ha conectado con la nube en el tiempo establecido, amarillo si se ha activado una alarma predefinida por el usuario, verde si todo está en

orden (se ha conectado en el tiempo establecido y no hay alarmas) y gris si el equipo está desconectado (figura 10).

Figura 10. Marcadores originales del visor de la web KunakCloud para cada estado de los dispositivos (verde, amarillo, rojo y gris).

Además, debido a la gran cantidad de marcadores que hay en el visor, hay implementado un sistema de clúster que agrupa los marcadores cuando se acumulan dentro de un radio predefinido. Estos clústeres toman el color del marcador más restrictivo que hay dentro del mismo. De tal forma que si hay un único marcador detectando algún inconveniente se pueda observar desde el clúster (figura 11).

Figura 11. Clústeres originales del visor de la web KunakCloud para cada estado del dispositivo más restrictivo en el interior (verde, amarillo, rojo y gris).

Los marcadores tienen la función de Popup, es decir cuando se les hace click encima aparece un cuadro en el que se muestra el nivel de batería, conectividad etc. del equipo. Además, al hacer click en uno de los equipos (tanto en el visor como en el menú de la izquierda) hace un zoom al marcador y aparece un nuevo panel de información en el lado derecho del visor (figura 12). En este panel aparecen los últimos valores de todos los parámetros medidos por el dispositivo en cuestión.

Figura 12. Dispositivo con popup y panel de información.

3. METODOLOGÍA

En esta sección se recogen brevemente las principales tecnologías y aplicaciones utilizadas en el desarrollo de este trabajo y se explica cómo se han desarrollado los objetivos del mismo.

3.1 TECNOLOGÍAS UTILIZADAS

Dado que durante el desarrollo de la metodología se harán referencias a tecnologías, software y lenguajes de programación de diferente índole, se resumen en este apartado para poder comprender con mayor claridad su función y utilización.

3.1.1 JavaScript

JavaScript (JS) es un lenguaje de programación interpretado, que permite dar funcionalidades a una página web. Es un lenguaje orientado a objetos, que aunque comparte muchas estructuras y características del lenguaje Java, fue desarrollado independientemente. Este lenguaje es opensource.

Se ha utilizado para el desarrollo del visor web además de la página web propiamente.

3.1.2 Eclipse

Eclipse es una plataforma de software que permite desarrollar entornos de desarrollo integrados (IDE). Se ha utilizado para simular una conexión cliente servidor en local, utilizando apache Tomcat. De esta forma se ha podido comprobar el código generado sin tener que publicarlo.

3.1.3 Leaflet

Leaflet es un plugin de JavaScript utilizado para realizar visores web. Leaflet es una alternativa más sencilla al ya conocido OpenLayers. Leaflet tiene una comunidad muy activa que genera plugin nuevos para esta herramienta, dotándola de más funcionalidades. Tal es así, que muchos de los plugin que se han utilizado en este trabajo son creados por la propia comunidad.

El visor web ya está generado mediante Leaflet, por lo tanto, en todas las modificaciones que se han realizado se ha utilizado esta herramienta.

3.1.4 Java Servlets

Los servlets son módulos escritos en Java que se utilizan en un servidor, en este caso un servidor web, permitiendo aumentar las potencialidades de la página web. Las posibilidades que brindan son muchas, de hecho originalmente, las coordenadas de los marcadores del visor, son obtenidas mediante un servlet que las busca en la base de datos y las envía al JavaScript. Se ha utilizado para añadir la funcionalidad de interpolación al visor.

3.1.5 R

R es un entorno de desarrollo de software libre para operaciones estadísticas y gráficas. Compila y ejecuta una gran variedad de plataformas UNIX, Windows y MacOS. Es un proyecto GNU similar al entorno y lenguaje de S, que fue producido por Bell Laboratories.

R proporciona una gran variedad de herramientas estadísticas y gráficas, teniendo además múltiples librerías y/o extensiones. Uno de los puntos más fuertes de este software es la gran calidad y versatilidad a la hora de crear material gráfico (Venables & Smith, 2018).

Se ha utilizado para realizar el script de interpolación.

3.1.6 Python

Python es un lenguaje de programación de alto nivel, orientado a objetos, con semántica dinámica. Combinando su alto nivel en estructura de datos y su escritura dinámica, es muy utilizado para Rapid Application (Desarrollo rápido de aplicaciones). Está diseñado tanto para ser un lenguaje de programación propiamente, como para ser un lenguaje capaz de unir y conectar otros entre sí. Python tiene numerosos módulos y paquetes, permitiendo programar en módulos y reutilizar código.

Se ha utilizado Python para unir los elementos del servlet, el código de R y ejecutarlo en Lambda (Guido van Rossum, 1991).

3.1.7 MySQL

MySQL es un sistema de gestión de bases de datos relacional de multi lectura, multi usuario utilizando SQL (Structure Query Language). Es una de las bases de datos de código abierto más utilizadas del mundo, especialmente en entornos de desarrollo web (MySQL AB, 1995).

Se ha utilizado para almacenar una copia de la base de datos original de la empresa y poder simular las consultas en local. Es la fuente de datos que abastece la página web y su visor.

3.1.8 GeoServer

GeoServer es un servidor de código abierto para intercambiar información geoespacial. Diseñado para la interoperabilidad, publica datos de casi cualquier fuente de datos utilizando open standards (OpenGeo, GeoSolutions, 2012).

Se ha utilizado para añadir capas mediante servicios WMS al visor.

3.1.9 AWS Lambda

AWS (Amazon Web Services) Lambda, es un servicio de informática sin servidores que ejecuta el código como respuesta a eventos como solicitudes HTTP. Lambda ejecuta el código en una infraestructura informática de alta disponibilidad y ejecuta la administración integral de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de

capacidad y el escalado automático, la implementación de seguridad y código, así como la monitorización y los registros. Lo único que hay que hacer es proporcionar el código (Amazon Web Services, 2014).

Se ha utilizado para ejecutar el código de R directamente desde el servidor.

3.2 BASE DE DATOS

A pesar de que no es uno de los temas que se ha planteado en los objetivos, es de vital importancia para todos ellos, el determinar si hay necesidad de implementar una base de datos espacial (por ejemplo, posgresql- postgis, o mysql con extensión gis). Esta implementación cambiaría la metodología de consultar la base de datos a la hora de mostrarlos en el visor, dado que, si existe una base de datos espacial, los elementos que se obtienen de la consulta pueden ser geometrías propiamente, siendo estas representables en un mapa, mientras que de lo contrario, es necesario hacer un tratamiento previo a los datos y crear las geometrías a partir de estos.

La implementación de una base de datos espacial trae consigo una serie de ventajas e inconvenientes que se analizarán a continuación:

Ventajas:

- Todos los datos están georreferenciados mediante coordenadas y se conoce el tipo de geometría que tienen (punto, línea, polígono, etc.) mediante el atributo de geometría. Esto permite realizar consultas de carácter espacial de forma sencilla que sería complicado de realizar con una base de datos relacional normal.
- Permite obtener la geometría de un punto, línea, o polígono y representarla directamente en un visor, o modificarlo mediante otro software sin tener que hacer transformaciones previas.

Inconvenientes:

- Necesidad de añadir la extensión espacial a la base de datos y cargar los datos en el nuevo formato con su atributo de geometría, o generar una nueva relación entre geometría y equipo.
- El personal de la empresa no tiene experiencia en el trabajo con datos espaciales ni de bases de datos espaciales, por lo que sería necesaria una formación.

Los objetivos planteados se ven afectados en diferente grado por esta decisión, en cuanto a los marcadores no se estima una gran afección debido a que ya están implementados en el visor y además al ser puntos, la adición de los mismos mediante leaflet es muy sencillo. Ocurre lo mismo para los clústeres. Sin embargo, la interpolación si se ve directamente afectada por esta decisión.

El objetivo de la interpolación es representar un área en la que se muestre la variación de una variable (un contaminante o la calidad del aire) a lo largo del espacio, partiendo de puntos con datos conocidos e interpolando/prediciendo los desconocidos. La representación de dicha interpolación puede ser realizada mediante imágenes ráster (mallas) georreferenciadas en formato tiff o geotiff, mostrando un valor por cada píxel, o mediante polígonos que mostrarían diferentes intervalos de la variable a representar mediante una rampa de colores.

El dilema no reside en el formato de representación, sino en la necesidad o no de almacenar el resultado. Es decir, si la funcionalidad de la interpolación generará un elemento a almacenar permanentemente cada vez que se utilice, o si esta representación será temporal y se borrará cuando se finalice. A priori no parece un problema excesivamente importante, pero si se genera un gran volumen de datos de este tipo, es muy recomendable (si no imprescindible) utilizar una base de datos espacial, mientras que en el caso opuesto podría añadirse el resultado al visor sin necesidad de almacenar la capa.

Teniendo los puntos anteriores en cuenta se ha optado por no establecer una base de datos espacial debido a que el objetivo no es generar capas de forma masiva sino mostrarlas cuando el cliente quiera mediante una funcionalidad del visor.

3.3 MEJORA DE LOS MARCADORES DE LOS DISPOSITIVOS

Como se ha comentado en el capítulo 2, los marcadores actuales del visor web contienen únicamente la ubicación de los dispositivos y un sistema de color que indica el estado de los mismos.

El primero de los objetivos de este trabajo consiste en mejorarlo de modo que aporte más información a simple vista. Dado que el sistema de color que hay implementado aporta gran información, se ha decidido mantenerlo y por tanto los cambios en los marcadores se han centrado en siguiente aspecto:

- **Mostrar información en el marcador**: se ha diseñado un nuevo marcador que muestra alguno de los parámetros del dispositivo dentro del marcador que lo representa.

Para ello se ha diseñado un nuevo marcador en forma de etiqueta, que muestra uno de los parámetros de los equipos que corresponden a cada marcador. El tamaño del marcador se ajusta al contenido de la etiqueta, es decir, cuantos más caracteres lleve la etiqueta más grande es el marcador y viceversa.

Hay que tener en cuenta que originalmente, la función de generar el icono es llamada por la función de generar el marcador ya implementada en el visor. Esta función utiliza llamadas asíncronas (Ajax) para obtener los datos de los dispositivos mediante un servlet, concretamente sus coordenadas (longitud y latitud), número de serie, tag, network, status y gateway. Estos marcadores son introducidos directamente en los clústeres como se muestra en el código del Anexo 1.

Es en las últimas líneas de código donde se introducen los parámetros a cada marcador incluyendo el icono. Este icono es generado por la función generateIconToMap() a la que se le pasan las propiedades del marcador anteriormente obtenidas.

Para realizar esta operación con leaflet, es necesario generar un marcador “html_marker” mediante un contenedor genérico “div”, es decir se genera un elemento html que sustituye al marcador original del que dispone el visor actualmente. Para ello se requiere de la librería original de leaflet y no es necesario ningún plugin para realizarlo. A pesar de ello, esto es realizable mediante otros plugin que existen para leaflet, puesto que existen numerosos plugin dedicados a modificar los marcadores que trae por defecto. El código utilizado para realizar esta operación es la que se muestra en el anexo Anexo 1.

La etiqueta tiene dos partes, una primera que es la flecha que apunta al lugar en el que se ubica en el mapa, mientras que la segunda parte es el cuadro donde se introduce el texto. Estas dos partes están contempladas en los ficheros de estilos de la página web y se introducen con las variables classTriangle y classRectangle. Ambas están diseñadas para que se les cambie el color en función del parámetro *status*.

Por defecto se ha preparado el marcador con las siguientes dimensiones:

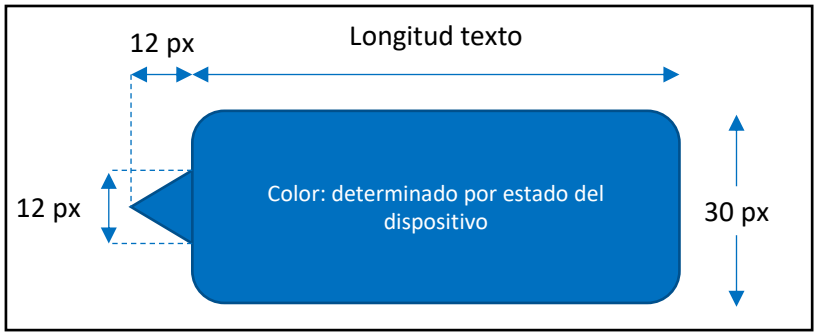


Figura 13. Dimensiones exteriores del nuevo marcador en píxeles.

Teniendo en cuenta la figura 13, para diseñar la etiqueta es necesario conocer el tipo de fuente y tamaño de los caracteres que estarán contenidos en ella, para así poder determinar el tamaño que tendrá la etiqueta. El problema llega cuando se pretende determinar la longitud (en píxeles) de esa cadena de caracteres que irá dentro de la etiqueta, puesto que no todos los caracteres ocupan lo mismo, imposibilitando la solución de establecer el tamaño en función del número de caracteres.

Para solventar este problema se ha creado una función dedicada a obtener el tamaño que necesita la etiqueta, que se llama al generar el icono propiamente. Esta función consiste en crear un elemento "ficticio" (puesto que no se incorporará al visor) mediante canvas, simulando un elemento html. En él se introducirá el texto que realmente irá en la etiqueta y puesto que es un elemento de html es posible conocer sus medidas en píxeles (figura 14). Una vez se determina esta longitud se introduce en el marcador.

```
function getWidthOfText(txt, fontname, fontsize){
    if(getWidthOfText.c === undefined){
        getWidthOfText.c=document.createElement('canvas');
        getWidthOfText.ctx=getWidthOfText.c.getContext('2d');
    }
    getWidthOfText.ctx.font = fontsize + ' ' + fontname;
    return getWidthOfText.ctx.measureText(txt).width;
}
```

Figura 14. Función creada con JavaScript, encargada de obtener la longitud de la etiqueta en función del contenido. Esta función es llamada por la función de generar los iconos de los marcadores.

Al generar la etiqueta y mostrarla en el visor, se crea un nuevo problema relacionado con el tamaño de la etiqueta y el nivel de zoom. Viendo las figuras anteriores, se puede apreciar cómo la etiqueta nueva es considerablemente más grande que los marcadores originales. Esto conlleva que cuando el zoom del mapa es muy bajo (es decir se ve más superficie terrestre y/o está más alejado), las nuevas etiquetas pasan a ocupar gran parte del visor tapándose entre sí.

Para solucionar este problema se ha escalado el tamaño de la etiqueta en función del nivel de zoom en el que se están representando utilizando un multiplicador en el tamaño. Esta operación se ha resuelto como se ve en la figura 15.

```
function generateMarkerHtml(status, ibValue,zoom){

                if (zoom<=3){
                    var multiplier = 0.25
                }
                else if (zoom <=7 ){
                    multiplier = 0.5
                }
                else if (zoom <=11){
                    multiplier = 0.75
                }else{
                    multiplier = 1
                }
```

Figura 15. Multiplicador de tamaño para el nivel de zoom dentro de la función de generar iconos de los marcadores.

Tal y como se ve en la figura anterior, este multiplicador es de 0.25, 0.5, 0.75 o 1 en función de si el zoom es mayor o igual que 3, mayor o igual que 7, mayor o igual que 11 o cualquier otro valor. Multiplicando este valor a la las dimensiones exteriores de la etiqueta ya queda resuelto el tamaño que ocupa en el visor. Sin embargo, queda otro punto importante por modificar: las dimensiones interiores.

Para poder albergar un texto, la etiqueta tiene ciertos márgenes interiores que permiten establecer caracteres en su interior sin que se salgan del mismo y quedando situados en el centro. Los parámetros a modificar son, el tamaño de letra, el padding y el margen superior.

Por defecto el tamaño de letra que tienen los marcadores es de 15px, pero si las dimensiones del marcador cambian el tamaño de letra tiene que hacerlo también. Se ha establecido que un tamaño de letra más pequeño que 8 no se aprecia bien en el visor, por lo que se altera con la siguiente expresión:

fontsize = Math.max(Math.round(15*multiplier),8)

Como se puede apreciar se aplica el multiplicador al tamaño original, si este valor es mayor que 8 se establece como nuevo tamaño de texto y en caso contrario se establece en 8 px.

En el caso del padding, para una altura de original de 30 px el padding es de 3 px. Por lo tanto se ha aplicado esta relación para todas las alturas, utilizando la siguiente expresión:

padding=Math.round(3*(height/30))

De esta forma, todos los tamaños de etiqueta mantienen la proporción original.

El margen superior es un tema importante en el marcador, debido a que es el encargado de que la etiqueta este centrada, es decir por defecto el rectángulo estaría alineado con el triángulo en la parte superior, pero para hacer que se alineen en el centro, se ha modificado el margen superior a -6px, haciendo que el rectángulo empiece más arriba que el triángulo haciendo coincidir sus centros.

Teniendo en cuenta la figura 13 donde se ven las dimensiones del rectángulo y del triángulo que forman la etiqueta, se puede observar cómo este margen corresponde a la mitad de la diferencia entre la altura del rectángulo y la del triángulo. Es decir, en el caso original este margen tiene un valor de -(30 -12)*0.5. Como las dimensiones del triángulo no cambian, su altura siempre será de 12 px, por lo que se puede establecer la siguiente expresión para calcular el margen superior de la etiqueta:

TopMargin =Math.round(-(height-12)/2)

Una vez que se han establecido las normas que rigen el tamaño de las etiquetas, es necesario que cada vez que se realice un zoom en el mapa, se vuelvan a generar de acuerdo al nuevo nivel de zoom. Para ello se ha creado un evento en el mapa encargado de refrescar los marcadores (figura 16).

```
map.on('zoomend',function(){
    if (map.getZoom()>=7){
    zoomlvl=false,
    refreshMarkers();
    }else {
    zoomlvl=true,
    refreshMarkers();}
})
```

Figura 16. Evento del mapa encargado de refrescar los marcadores cada vez que se realiza zoom.
Nota: el código de esta figura muestra más funcionalidades que las mencionadas destinadas a los clústeres.

3.4 MODIFICACIÓN DE LA VISUALIZACIÓN DE LOS CLÚSTERES

En este apartado se resume cómo se ha procedido con la modificación de los clústeres para completar los objetivos establecidos en el apartado 2. Estos objetivos no son otros que mejorar la visualización de los clústeres.

Este apartado está muy ligado al anterior puesto que el clúster agrupa las etiquetas cuando están dentro de determinado rango, generando un marcador diferente que muestra el número de etiquetas que alberga. El sistema de clúster que está implementado originalmente, hace que el clúster tome el color más restrictivo entre los dispositivos que alberga. Partiendo de este punto, la visualización se puede mejorar de dos formas:

- Alternativa 1: simplificar la representación del mapa mostrando únicamente clústereres o únicamente marcadores.
- Alternativa 2: Mostrar en el clúster más información sobre los dispositivos que albergan dentro.

Alternativa 1:

Esta alternativa consiste en mantener los clústeres originales, pero modificando el parámetro llamado “singleMarkerMode”, haciendo que los marcadores que estén aislados se conviertan en clústeres de tamaño 1 y eliminando las etiquetas. El inconveniente de esta alternativa es que una vez se ha establecido este parámetro se sustituye el marcador original por el del clúster y por tanto al hacer zoom sigue quedándose el símbolo del clúster.

Para evitar este problema, se le asigna el control de este parámetro a una variable que estará en TRUE mientras el nivel de zoom sea bajo (el visor esté lejos) y se volverá false cuando se llegue a determinado nivel de zoom (se ha establecido zoom 7 como umbral).

Para realizar el cambio de tamaño de los marcadores ya se ha contemplado la necesidad de refrescarlos cada vez que se hace zoom, por lo que al mismo tiempo se actualiza la visualización o no de los clústeres individuales. El código para realizar esto se muestra en el Anexo 1.

Alternativa 2

Esta alternativa consiste en cambiar los clústeres originales, por un nuevo tipo de clúster generado utilizando el pluging para leaflet "Prune_Cluster" (Antoine Pultier "yellowiscool," 2014). Este nuevo clúster, permite modificar la apariencia del clúster por una que en el centro del circulo muestra el número de marcadores que alberga, pero en la circunferencia exterior se muestra un porcentaje de la cantidad de dispositivos que hay en cada estado en forma de gráfico circular.

Para llevarlo a cabo se ha creado primero el icono que llevará el clúster extendiendo el método L.icon de leaflet. Esto permite generar un icono personalizado que utilizará el clúster. El código utilizado para generar este nuevo icono se muestra en Anexo 1.

Esta función permite crear un icono que en el centro muestra un circulo blanco con un número negro indicando el número de marcadores dentro, mientras que en el borde del círculo se muestra la proporción de marcadores de cada color que hay.

Posteriormente se genera el clúster, que llama a la función anterior para crear el icono (figura 17).

```
ClusterPrune = new PruneClusterForLeaflet(20,2);

ClusterPrune.BuildLeafletClusterIcon = function(cluster) {
var e = new L.Icon.MarkerCluster();
e.stats = cluster.stats;
e.population = cluster.population;


return e;
};
```

Figura 17. Función de crear el PruneCluster. El código completo está en el Anexo 1

Finalmente, solo queda añadir los marcadores al clúster creado. La diferencia con el clúster original es que la longitud y la latitud no son partes de una misma lista al introducirlos, sino que son parámetros independientes del propio marcador. Esto imposibilita la operación getLngLat() que tiene leaflet para obtener las coordenadas de un punto. Para solventar este problema se generan ambos tipos de marcador, los PruneCluster.Marker para representar en el mapa y los anteriores para poder hacer este tipo de consultas. Este clúster no tiene la opción de dibujar un clúster de tamaño 1 en el visor por lo que no es posible implementarlo.

3.5 AÑADIR LA FUNCIONALIDAD DE INTERPOLACIÓN AL VISOR

En este apartado se pretenden resumir los pasos seguidos para conseguir uno de los objetivos planteados al inicio. Este objetivo trata de añadir la funcionalidad de interpolar los valores que muestran los marcadores, en un área escogida por el usuario en el visor.

Esta operación se divide en 4 fases diferentes, que difieren en la última, la forma de mostrar el resultado en el visor generando 3 alternativas:

- Generar el polígono donde se realizará la interpolación
- Enviar el polígono y los dispositivos desde el visor
- Realizar la interpolación
- Mostrar el resultado (3 alternativas)
    - Introducir la imagen como ráster
    - Poligonizar el resultado
    - Generar líneas de corte

Por tanto, se resumen primero las partes que son comunes a todas las alternativas y posteriormente cada una de las posibles soluciones.

3.5.1 Generar el polígono donde se realizará la interpolación

El primer paso para poder realizar la interpolación es añadir la funcionalidad de generar un polígono. Se ha generado un botón en la parte superior izquierda del visor para poder empezar a dibujar en el mapa. Al contrario que OpenLayers, Leaflet no tiene un control para añadir la funcionalidad de dibujar al visor, por lo que es necesario generarlo. Para ello, se ha creado una función en que será llamada en 3 instancias una vez se empieza a generar el polígono. Para activar la función de dibujar, se utiliza el anteriormente citado botón, introducido mediante el plugin easyButton de leaflet (figura 18).

```
L.easyButton('fa-globe', function(btn, map){

    cuadro = function(e){
        var latlng = map.mouseEventToLatLng(e.originalEvent)
        elcuadro(2,latlng)
    }
    clickar2 = function(e){
        var latlng = map.mouseEventToLatLng(e.originalEvent)
        map.off('mousemove', cuadro)
        map.off('click',clickar2)
        elcuadro(3,latlng)
        puntoCuadro=[]
    }
    clickar = function(e) {
        var latlng = map.mouseEventToLatLng(e.originalEvent)
        map.on('mousemove', cuadro);
        map.on('click',clickar2);
        map.off('click',clickar);
        elcuadro(1,latlng);
        }
    //map.off('click', doStuff)
    map.on('click', clickar)


}).addTo(map);
```

Figura 18. Código de JavaScript que habilita el botón para poder dibujar un rectángulo en el mapa.

Al hacer click en el nuevo botón, se añade al visor un evento que saltará al pinchar en un punto del mapa. Cuando este evento ocurra, llamará a la función de generar el polígono (llamado elcuadro()) indicando que es la primera instancia y aportado las coordenadas del punto. La función almacena estas coordenadas como las primeras de una lista. Este click además deshabilita el evento de click del mapa, y habilita otros dos: uno que saltará cada vez que el ratón se mueva, y otro nuevo que se activará al volver a hacer click.

Cada vez que el ratón se mueva se llamará a la función de generar el polígono indicando que es la segunda instancia y pasándole las coordenadas del movimiento. La función genera un rectángulo desde el punto en el que se hizo el primer click hasta la posición actual del puntero. Esta parte es la encargada de que el usuario pueda ver en todo momento como está generando el polígono en tiempo real.

Finalmente, cuando se ejecuta el segundo click, se llama a la función indicando que es la tercera instancia y pasando las coordenadas finales (figura 19).

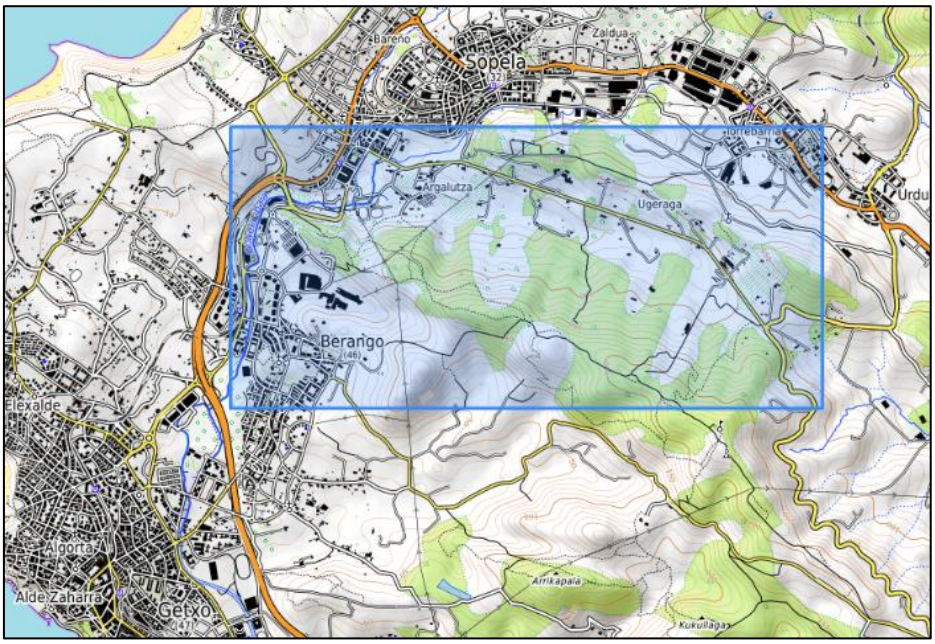


Figura 19. Ejemplo de dibujo del polígono sobre el visor.

La función de “elcuadro” necesita saber la instancia o turno en el que se llama y las coordenadas del punto que ha recogido ese evento. El turno uno almacena la primera coordenada del polígono al hacer click, el segundo lo va actualizando mientras se mueve el ratón y el tercero genera el polígono final. Una vez generado el polígono se determina qué dispositivos son los que están dentro del mismo, ya que los valores utilizados en la interpolación derivarán de estos. Una vez obtenidos los dispositivos, se lanza la función de interpolación llamado callHeatMap pasándole el polígono y los ID de los dispositivos como parámetros. El código utilizado para realizar esta operación se muestra en el anexo 1.

3.5.2 Enviar el polígono y los dispositivos desde el visor

La función callHeatMap() utiliza Ajax para llamar a un servlet preparado para realizar la interpolación. La particularidad de este tipo de función es, que la aplicación no permanece en espera hasta que termina la función, sino que puede seguir siendo utilizada mientras la función realiza su trabajo. Esto permite al usuario seguir utilizando el visor y la página web mientras se está cargando la interpolación, que será añadida cuando termine sin haber interrumpido la labor del usuario.

Una vez los datos han llegado al servlet, hace una consulta a la base de datos para determinar qué dispositivos llevan GPS y cuáles no. Posteriormente el servlet envía a lambda AWS en formato texto, las coordenadas del polígono, los dispositivos con GPS, los dispositivos sin GPS y las fechas de la interpolación. El código encargado de ejecutar esta operación se muestra en el Anexo 1.

Se han omitido ciertas líneas de código correspondientes a credenciales para la base de datos y los servicios web de Amazon.

3.5.3 Ejecutar la interpolación

Para que lambda AWS pueda realizar la interpolación, es necesario subir a dicho servidor el script que tiene que utilizar. En este caso es un script de Python en el que va embebido código de R. Es decir, dentro del propio script de Python está introducido como texto un script de R, encargado de realizar la interpolación. El trabajo de Python es recibir los datos del servlet introducirlos en el código de R (Python tiene un módulo que le permite trabajar con código R), para que R posteriormente lleve a cabo la interpolación y devuelva el resultado a Python, que lo envía a su vez al servlet.

En resumen, Python ejerce de comunicador entre el servlet y lambda AWS y R es el encargado de hacer la interpolación.

Una vez que los datos enviados por el servlet llegan a R, este se encarga de dar un formato válido para R a los datos de entrada. Posteriormente, realizar la consulta a la base de datos, obteniendo los valores del contaminante para los dispositivos en la fecha establecida. Esta consulta difiere en función de la existencia o ausencia de GPS en los dispositivos, de ahí su distinción. Una vez obtenidos los datos del contaminante de los dispositivos y sus coordenadas, se interpolan en área del polígono generando un ráster.

3.5.3.1 Modelo de interpolación

El modelo que se ha utilizado en la interpolación es un modelo geoestadístico lineal basado en estimación de máxima verosimilitud (MLE). Cada vez que se ejecuta la interpolación, los datos que se reciben para realizarla son muy diferentes, es por eso que no es posible ajustar un modelo previamente a los datos de entrada, sino que sería necesario ajustar un modelo cada vez que se ejecute la interpolación.

Hay que tener en cuenta que lo que se busca con esta interpolación no es dar un resultado exacto al usuario, sino que pueda hacerse una ligera idea de cómo se distribuye el parámetro medido en el

espacio que ha seleccionado. Es por ello que en este caso se prima la sencillez de aplicación, a la precisión de los resultados.

Tal y como está diseñado el proceso, es muy complicado realizar este ajuste de modelos y de ahí que el modelo elegido sea el anteriormente citado. Además la implementación en R del mismo es muy sencillo mediante el paquete ”geostatsp” (Patrick Brown, 2018).

3.5.4 Mostrar el resultado

3.5.4.1 Introducir la imagen como ráster

Para esta solución es necesario utilizar GeoServer, en el que se guarda el resultado y se muestra en el visor como un servicio WMS. Esa solución requiere introducir GeoServer en el servidor y almacenar ahí los resultados. Leaflet cuenta con la opción de introducir capas mediante WMS, por lo que es una opción válida.

3.5.4.2 Poligonizar el resultado

R cuenta con la opción de poligonizar el raster, generando polígonos entre los rangos de cantidad de contaminante establecidos. Se pueden crear tantos rangos/polígonos como se quiera, con la ventaja de poder generar un Json con ellos enviándolos al servlet. Cuando el servlet envía el Json al JavaScript, este lo recibe en la llamada Ajax que había realizado. Leaflet tiene la opción de añadir directamente al visor geometrías que llegan en formato Json, por lo tanto, se asigna un color a cada polígono en función del valor del contaminante y se añade al visor.

3.5.4.3 Generar líneas de corte

Está opción es parecida a la anterior, con la diferencia de que en este caso, en lugar de generar polígonos a partir del ráster, se generan líneas de contorno como si fueran curvas de nivel. Se puede generar un Json con estas líneas y enviarlas al servlet de la misma forma que en el caso anterior. Esto permite añadir al visor las líneas directamente, o generar los polígonos en el propio JavaScript partiendo de las coordenadas de las líneas. La ventaja de este método frente al anterior, es que reduce el tiempo en el que se lleva a cabo el proceso debido a que poligonizar es una función muy lenta que en los ensayos ha costado en torno a 35 segundos frente a 7 que cuesta el script de líneas de contorno.

3.6 OTRAS MEJORAS

Este apartado está dedicado a ciertas funcionalidades o complementos que inicialmente no forman parte de los objetivos, pero han sido desarrolladas puesto que mejoran la funcionalidad o calidad del visor.

3.6.1 Buscador en el visor

Como se ha comentado anteriormente en los antecedentes, se puede observar como hay una columna a la izquierda del visor, dedicada a listar los dispositivos que están desplegados. Con el objetivo de

ahorrar espacio en la página web, se ha añadido un botón de filtrado de dispositivos dentro del propio visor. En cuanto se hace click en el botón se genera un espacio para escribir, que al introducir cualquier carácter filtra todos aquellos dispositivos que empiecen por el mismo listándolos. Además, se puede seleccionar, tanto con el ratón como con el teclado, el dispositivo deseado haciendo zoom al mismo y marcándolo en el visor con un círculo rojo. Para introducirlo se ha utilizado el plugin searchControl de leaflet (Cudini, 2013).

Tanto la opción de hacer zoom al seleccionar como la de marcarlo con un círculo son opcionales y pueden ser alteradas (escogiendo el nivel de zoom deseado) o desactivadas desde el JS (figura 20).

```
searchControl = new L.Control.Search({
        layer: markerClusters,
        propertyName: 'title',
        zoom: 17,
        circleLocation: true
}).addTo(map);
```

Figura 20. Código JavaScript para añadir el control de búsqueda en el visor.

3.6.2 Mini mapa

El minimapa consiste en añadir otro mapa al visor, pero de tamaño significativamente más pequeño. Se puede ubicar en cualquier parte del mismo, aunque por defecto se ubica en la esquina inferior derecha. En él se muestra la misma zona que se ve en el mapa principal, pero con un zoom alejado de tal forma que, si se ha llegado a un marcador seleccionándolo en el buscador y haciendo zoom automático, se pueda ubicar mirando el minimapa.

Es muy personalizable, aunque trae muchos parámetros por defecto que no es necesario alterar (por ejemplo, el nivel de zoom o el mapa de fondo). Para introducirlo se ha utilizado el plugin mini map de leaflet (Nordan, 2012)(figura 21).

```
new L.Control.MiniMap(L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'), {
toggleDisplay: true
}).addTo(map);
```

Figura 21. Código de JavaScript para añadir el control de minimapa al visor.

3.6.3 Marcadores de calidad del aire WAQI

Utilizando la API de WAQI air quality se han introducido marcadores de calidad del aire en el visor web. De esta forma se puede apreciar con una rápida ojeada la calidad de aire que hay en el entorno.(Quality, 2008). Esto permite mostrar datos de calidad del aire de otra fuente permitiendo contrastarlos con los existentes en el mapa. Es necesario ponerse en contacto por correo para recibir el "token" de acceso. El código utilizado es el siguiente (figura 22):

```
var WAQI_URL    = "https://tiles.waqi.info/tiles/usepa-aqi/{z}/{x}/{y}.png?token=*******";
var WAQI_ATTR   = 'Air Quality Tiles &copy; <a href="http://waqi.info">waqi.info</a>';
var waqiLayer   = L.tileLayer(WAQI_URL, {attribution: WAQI_ATTR});
```

Figura 22. Código para WAQI Layer

4. RESULTADOS Y DISCUSIÓN

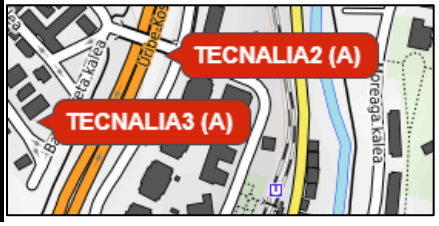
En este apartado se describen los resultados de la metodología adoptada en cada situación. En alguno de los apartados existe más de una opción metodológica que se ha llevado a cabo, pero aquí se describe únicamente la que mejor se ajusta a las necesidades iniciales y se justifica su elección.

4.1 MARCADORES

Solo se ha contemplado una solución para los marcadores que es la que se ha implementado en el visor. Dicha solución trata de marcadores en forma de etiquetas que muestran uno de los valores medidos. Las etiquetas cuentan con el sistema de color implementado, cambiando el color en función del estado del dispositivo (gris si está desconectado, rojo si no se ha conectado en el tiempo establecido, amarillo alarma y verde correcto). Además, los marcadores modifican su longitud en función del texto a introducir y alteran el tamaño de letra y altura en función del nivel de zoom.

En la siguiente Tabla 1 se resumen los resultados obtenidos para cada umbral de nivel de zoom.

Tabla 1. Resumen de los resultados obtenidos en la visualización de marcadores para cada umbral de nivel de zoom establecido.

Nivel de zoom	Resultado
<=3	
<=7	
<=11	
>11	

4.2 CLÚSTERES

Se han presentado dos soluciones diferentes en este apartado, una manteniendo los clústeres originales, pero sustituyendo las etiquetas individuales por clústeres de tamaño 1, mientras que la segunda opción sugería un cambio de apariencia en los clústeres indicando en el borde la cantidad de dispositivos de cada estado que albergan (figura 23).

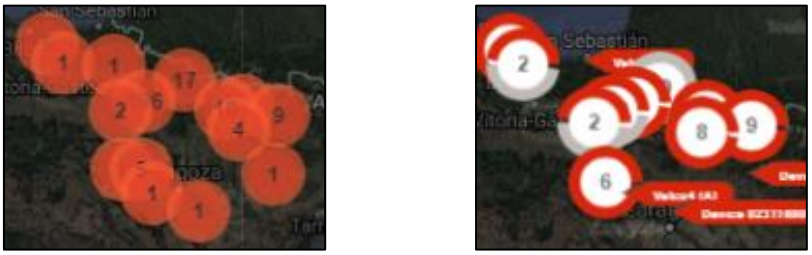


Figura 23. Comparativa resultados de los clústeres, alternativa 1 (izquierda) frente a la alternativa 2 (derecha). NOTA: los colores que aparecen en las imágenes son rojo y gris debido a que la copia almacenada en local es antigua y por lo tanto los datos no están actualizados.

La propuesta de la primera solución se crea debido a la gran cantidad de espacio que ocupan los nuevos marcadores, de tal forma que en un zoom lejano dificultan la visualización de los elementos del mapa. Sin embargo, al haber implementado el cambio de dimensiones en función del nivel de zoom en las etiquetas, este problema queda solventado.

Por lo tanto, la opción que se ha añadido es la segunda, que aporta un valor añadido al poder detectar la proporción de etiquetas de cada tipo que hay en el interior, observando únicamente el clúster.

4.3 INTERPOLACIÓN

La interpolación es el apartado más complejo de todos debido a la gran cantidad de alternativas que se han generado y probado. Todas ellas han demostrado ser funcionales, pero algunas de ellas son de implementación más sencilla, bien requieren menos recursos en ejecución o generan resultados de mayor calidad.

Para poder determinar realmente cuál de ellas implementar, se ha realizado un análisis de alternativas, evaluando la facilidad de implementación, velocidad de ejecución y la calidad del resultado obtenido. Estos parámetros se han valorado en una escala del 1 al 5, siendo 1 el peor valor y 5 el mejor. Por otra parte, a cada parámetro se le ha asignado un peso de modo que se ha realizado una calificación final ponderando todos los resultados, 0,5 para la facilidad de implementación, 0,25 la velocidad de ejecución y 0,25 para la calidad del resultado final. El análisis de alternativas se resume en la Tabla 2.

Tabla 2. Análisis de alternativas para la visualización del resultado de interpolación.

Alternativa	Facilidad de implementación (0.5)	Velocidad de ejecución (0.25)	Calidad del resultado (0.25)	Resultado final
Lambda/Geoserver/ráster	1	5	5	3
Lambda/polígonos	4	2	3	3.25
Lambda/líneas	4	3	2	3.25

Para entender mejor los valores mostrados en la tabla anterior, se explicará con detalle el porqué de la valoración.

La ponderación que se ha utilizado viene marcada diferentes motivos. La facilidad de implementación tiene más peso que el resto, porque el margen de tiempo que hay para llevar a cabo el trabajo es limitado y por tanto las opciones más sencillas son más apropiadas. En cuanto a la velocidad de ejecución y calidad del resultado, son dos cosas que van reñidas, normalmente lo que cuesta más tiene una mayor calidad y viceversa, es por ello que se les ha dado el mismo peso.

En cuanto a la facilidad de implementación se refiere, introducir el resultado directamente desde Lambda no requiere de utilizar un software adicional como geoserver, el cuál sería necesario implementar y posteriormente automatizar la publicación de datos en formato WMS. Por lo tanto, en este sentido introducir los resultados directamente obtenidos desde Lambda aporta mucha ventaja.

En cuanto a la velocidad de ejecución, se explica por sí solo, es la rapidez con la que se muestra el resultado en el visor. Este parámetro depende directamente del código de R, siendo muy rápido a la hora de generar un ráster, pero no tanto cuando los poligoniza. Es por eso que los puntos se han repartido de esa manera, siendo aquellas opciones que dependen de polígonos las más perjudicadas mientras que las que no lo necesitan tienen un mejor resultado.

Finalmente se valora la calidad del resultado, en este aspecto sin duda el introducir un ráster es la opción ideal debido a que se puede ver el gradiente entero de colores a lo largo de la imagen, sin embargo, la opción de generar polígonos para cada intervalo tampoco es una opción mala. No obstante, dichos polígonos son generados a partir un ráster y por tanto la poligonización de este ráster genera polígonos con bordes cuadriculados, lo cual no es estéticamente óptimo. Por último, las líneas mostrarían intervalos al igual que en el caso de los polígonos, pero no es visualmente tan atractivo.

Las alternativas generan los siguientes resultados en el visor (Figuras 24 y 25):

Figura 24. Interpolación introducida mediante polígonos (izquierda). Interpolación introducida mediante líneas (derecha)

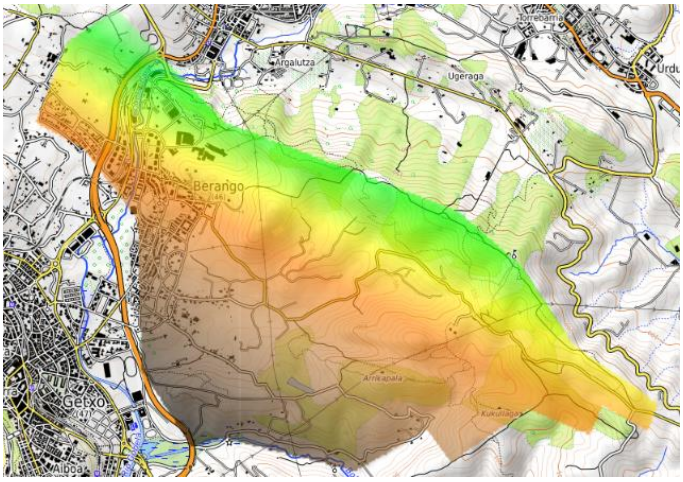


Figura 25. Interpolación introducida como ráster mediante GeoServer.

Dicho esto, se determina que la opción óptima de implementación es la de utilizar Lambda para generar polígonos e introducirlos en el visor directamente. En cuanto al botón que permite ejecutar la interpolación se ha utilizado el siguiente icono situado en la parte superior izquierda del visor (Figura 26).

Figura 26. Botón para generar el polígono y ejecutar la interpolación.

4.4 OTRAS MEJORAS

Tanto el minimapa como el buscador han resultado ser realmente útiles dentro del visor. En especial el buscador (Hu, Xiao, & Ishikawa, 2018), debido a que aporta un gran valor añadido, pudiendo reservar la columna izquierda de la página web para otros usos, como podrían ser nuevas herramientas para el visor o la página web.

Además, el hecho de que al buscar un dispositivo en el buscador lleve al usuario hasta el mismo, le aporta dinamismo al visor, que es complementado con el minimapa para reubicarse rápidamente (Figura 27).

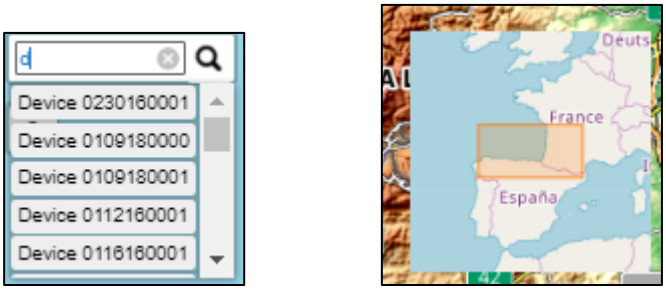


Figura 27. Herramienta de búsqueda (izquierda) y herramienta de minimapa (derecha).

En cuanto a capa de puntos con calidad de aire mundial, tiene el aspecto que se muestra en la Figura 28. Esta herramienta permite obtener información adicional de otra fuente que se puede contrastar con los datos de los sensores que dispone el usuario, dando mayor veracidad a los datos representados. Los marcadores cambian de color en función de si es un valor aceptable (verde), hay que tener cuidado (amarillo) o la calidad del aire es mala o perjudicial (rojo).

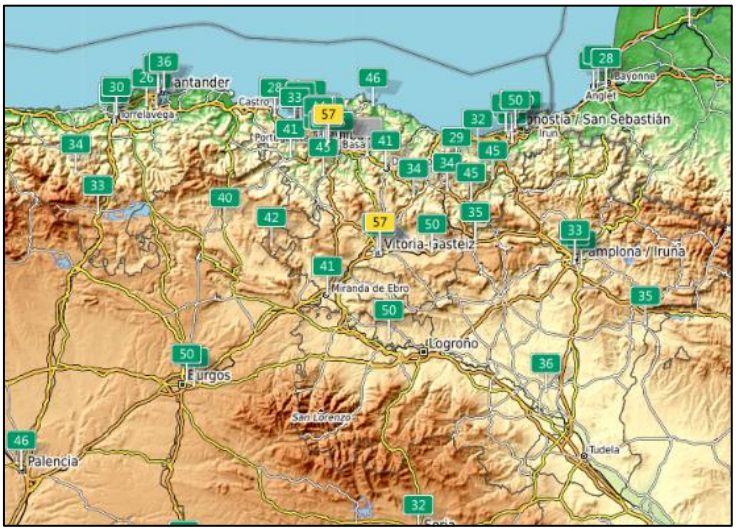


Figura 28. Aplicación de la API de AQI en el visor.

5. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se comienza comentando las conclusiones obtenidas tras la realización del trabajo:

- Leaflet ha demostrado ser una herramienta útil a la hora de crear visores web y dotarlo de funcionalidades.
- Las mejoras añadidas al visor son realmente útiles para un usuario que quiere conocer el estado de sus dispositivos de forma rápida. Añadiendo la funcionalidad de interpolar, se puede observar fácilmente el estado aproximado del entorno.
- Tanto el minimapa como el buscador han resultado ser realmente útiles dentro del visor. El buscador, además de dar un valor añadido al visor, permite reservar la columna izquierda de la página web para otros usos, como podrían ser nuevas herramientas para el visor o la página web.

  Además, el hecho de que al buscar un dispositivo en el buscador, lleve al usuario hasta el mismo, le aporta al visor un dinamismo que es complementado con el minimapa para reubicarse rápidamente.
- Una aplicación web bien estructurada y orientada hacia el usuario es de gran ayuda en la toma de decisiones aportando información de forma clara y práctica, como bien se indica en (Giannaros et al., 2018) donde se describe la utilización una aplicación de web mapping para monitorizar el viento en las olimpiadas de Brasil 2016.
- Se ha demostrado que se pueden añadir múltiples funcionalidades a un visor web que permiten realizar operaciones sencillas igual que un gis de escritorio, con la particularidad de estar en una página web, sin consumir tantos recursos del usuario.
- La utilidad de estos visores permite poner en valor todo el trabajo que tiene por detrás una empresa. Es decir, en este caso, el hecho de poder mostrar tan gráfica y claramente los datos obtenidos por los equipos en una página web, es lo que hace que la información generada a través de la actividad diaria sea útil y reconocida.

Por último, se comentan las actuaciones que podrían complementar el trabajo realizado:

- En cuanto a la interpolación existe una línea de trabajo por desarrollar, basado en utilizar un modelo más ajustado a los datos. Dado que se están midiendo contaminantes en la atmósfera, un modelo como el de Aermod (EPA, 2015) podría funcionar mejor que el actualmente utilizado. A pesar de ello, dado que la función de añadir el resultado al visor ya está implementada, con cambiar el script encargado de realizar la predicción sería suficiente para llevar esta fase a cabo.
- Otro punto de mejora o avance, sería el optimizar el tiempo de ejecución de la interpolación. En este aspecto se podría tender a poligonizar las líneas generadas en el propio JavaScript o encontrar una alternativa a la poligonización del ráster en R.

6. REFERENCIAS

Amazon Web Services. (2014). AWS Lambda – Características del producto. Retrieved September 13, 2018, from https://aws.amazon.com/es/lambda/features/

Antoine Pultier "yellowiscool." (2014). PruneCluster. Retrieved from https://github.com/SINTEF-9012/PruneCluster

Cudini, S. (2013). leaflet-search. Retrieved from https://github.com/stefanocudini/leaflet-search

Eanes, F. R., Silbernagel, J. M., Hart, D. A., Robinson, P., & Axler, M. (2018). Participatory mobile- and web-based tools for eliciting landscape knowledge and perspectives: introducing and evaluating the Wisconsin geotools project. *Journal of Coastal Conservation*, *22*(2), 399–416. https://doi.org/10.1007/s11852-017-0589-2

Elwood, S., & Leszczynski, A. (2013). New spatial media, new knowledge politics. *Transactions of the Institute of British Geographers*, *38*(4), 544–559. https://doi.org/10.1111/j.1475-5661.2012.00543.x

EPA (Environmental Protection Agency). (2015). Revision to the guideline on air quality models: Enhancements to the AERMOD dispersion modeling system and incorporation of approaches to address ozone and fine particulate matter. *Federal Register*, *80*(145), 45339–45387.

Giannaros, T. M., Kotroni, V., Lagouvardos, K., Dellis, D., Tsanakas, P., Mavrellis, G., … Vakkas, T. (2018). Ultrahigh resolution wind forecasting for the sailing events at the Rio de Janeiro 2016 Summer Olympic Games. *Meteorological Applications*, *25*(1), 86–93. https://doi.org/10.1002/met.1672

Guido van Rossum. (1991). What is Python? Executive Summary | Python.org. Retrieved September 13, 2018, from https://www.python.org/doc/essays/blurb/

Hare, T. M., Rossi, A. P., Frigeri, A., & Marmo, C. (2018). Interoperability in planetary research for geospatial data analysis. *Planetary and Space Science*, *150*, 36–42. https://doi.org/10.1016/j.pss.2017.04.004

Hu, S., Xiao, C., & Ishikawa, Y. (2018). An efficient algorithm for location-aware query autocompletion. *IEICE Transactions on Information and Systems*, *E101D*(1), 181–192. https://doi.org/10.1587/transinf.2017EDP7152

Knoth, C., Slimani, S., Appel, M., & Pebesma, E. (2018). Combining automatic and manual image analysis in a web-mapping application for collaborative conflict damage assessment. *Applied Geography*, *97*, 25–34. https://doi.org/10.1016/j.apgeog.2018.05.016

Kunak. (2018). Kunak Technologies - Sobre nosotros. Retrieved September 14, 2018, from https://www.kunak.es/empresa/sobre-kunak/

Martin, D. G. (2004). Nonprofit Foundations and Grassroots Organizing: Reshaping Urban Governance. *The Professional Geographer*, *56*(3), 394–405. https://doi.org/10.1111/j.0033-0124.2004.05603008.x

Melis, M. T., Dessì, F., Loddo, P., La Mantia, C., Da Pelo, S., Deflorio, A. M., … Mwasi, B. N. (2018). Improving land cover mapping: A mobile application based on esa sentinel 2 imagery. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*

- *ISPRS Archives* (Vol. 42, pp. 1263–1266). https://doi.org/10.5194/isprs-archives-XLII-3-1263-2018

MySQL AB, S. M. y O. C. (1995). MySQL Documentation. Retrieved September 13, 2018, from https://dev.mysql.com/doc/

NavarraCapital.es. (2018). Kunak logra 1,6 M€ de la fase II del programa Horizonte2020 | Navarra Capital. Retrieved September 13, 2018, from https://navarracapital.es/kunak-logra-16-me-de-la-fase-ii-del-programa-horizonte2020/

Nordan, R. (2012). Leaflet-MiniMap. Retrieved from https://github.com/Norkart/Leaflet-MiniMap/

OpenGeo, GeoSolutions, R. R. (2012). GeoServer. Retrieved September 13, 2018, from http://geoserver.org/

Patrick Brown, R. H. (2018). Geostatistical Modelling with Likelihood and Bayes. https://doi.org/10.18637/jss.v063.i12>.License

Quality, W. A. (2008). API - Air Quality Programmatic APIs. Retrieved from https://aqicn.org/api/es/

Sack, C. M. (2018). The Status of Web Mapping in North American Higher Education, (89), 25–43. https://doi.org/10.14714/CP89.1429

Savini, L., Tora, S., Di Lorenzo, A., Cioci, D., Monaco, F., Polci, A., … Conte, A. (2018). A web geographic information system to share data and explorative analysis tools: The application to West Nile disease in the Mediterranean basin. *PLoS ONE*, *13*(6). https://doi.org/10.1371/journal.pone.0196429

Venables, W. ., & Smith, D. M. (2018). An Introduction to R. *A Programming Environment for Data Analysis and Graphics*, *57*(C), 1–40. https://doi.org/10.1016/B978-0-12-381308-4.00001-7

Wang, L., & Cheng, Q. (2008). Flood forecasting and risk mapping using a web-based spatial decision support services approach. In *International Geoscience and Remote Sensing Symposium (IGARSS)* (Vol. 3). https://doi.org/10.1109/IGARSS.2008.4779415

Universidad Pública de Navarra

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS AGRONOMOS

Nafarroako Unibertsitate Publikoa

NEKAZARITZAKO INGENIARIEN

GOI MAILAKO ESKOLA TEKNIKOA

TRABAJO FIN DE MÁSTER

MEJORA E IMPLEMENTACIÓN DE NUEVAS FUNCIONALIDADES A UN VISOR EN LA EMPRESA KUNAK TECHNOLOGIES S.L.

ANEXO 1: Código de programación

Presentado por:

Iban Iturria Aguinaga

Máster Universitario en SIG y Teledetección

ÍNDICE

1. CÓDIGO DE JAVASCRIPT

1.1 CÓDIGO PARA DETERMINAR LA LONGITUD DEL MARCADOR

Esta función es la encargada de determinar la longitud que tendrá el marcador. Se le introducen como parámetros el texto, el nombre de la fuente y el tamaño de la misma y genera un elemento ficticio en el que lo introduce. Finalmente devuelve la longitud de este elemento.

```
function getWidthOfText(txt, fontname, fontsize){
    if(getWidthOfText.c === undefined){
        getWidthOfText.c=document.createElement('canvas');
        getWidthOfText.ctx=getWidthOfText.c.getContext('2d');
    }
    getWidthOfText.ctx.font = fontsize + ' ' + fontname;
    return getWidthOfText.ctx.measureText(txt).width;
}
```

1.2 CÓDIGO PARA GENERAR LOS NUEVOS MARCADORES

Esta función es la encargada de generar los marcadores, le llegan como parámetros el estado del dispositivo, el texto "ibValue" y el nivel de zoom. Lo primero que hace es escoger un multiplicador en función del nivel de zoom, que luego se aplicará a las dimensiones del marcador.

Después de aplicar el multiplicador a las medidas exteriores conocidas y el tamaño de fuente, llama a la función anteriormente mencionada para obtener la longitud del marcador. Finalmente genera el HTML que será el nuevo marcador, introduciendo las medidas internas relacionadas con el nivel de zoom (paddig, top margin) y lo devuelve.

```
function generateMarkerHtml(status, ibValue,zoom){

                if (zoom<=3){
                    var multiplier = 0.25
                }
                else if (zoom <=7 ){
                    multiplier = 0.5
                }
                else if (zoom <=11){
                    multiplier = 0.75
                }else{
                    multiplier = 1
                }

                var classTriangle = status===3?"triangle-left-
hearbeat":((status===2)?"triangle-left-alarms":((status===0)?"triangle-
left-ok":"triangle-left-shutdown"));
                var classRectangle = status===3?"rectangle-
hearbeat":((status===2)?"rectangle-alarms":((status===0)?"rectangle-
ok":"rectangle-shutdown"));
                var fontsize = Math.max(Math.round(15*multiplier),8)
                var height = Math.round(30*multiplier)
```

```
                var longitud=Math.round(getWidthOfText("
"+String(ibValue)+" (A)",("Helvetica Neue",' Arial', 'Helvetica', 'sans-
serif'),fontsize+'px'))+30;

                return "<div
style='height:"+height+"px;width:"+longitud+"px;display: flex;'>"
                +"<div class='"+classTriangle+"'
style='float:left;'></div>"
                +"<div class='"+classRectangle+"' style='font-size:
"+fontsize+"px;margin-top: "+String(Math.round(-(height-
12)/2))+"px;border:0.5px solid #4e4a4a;border-radius: 9px 10px 10px
9px;height:"+height+"px;
width:"+longitud+"px;overflow:hide;padding:"+String(Math.round(3*(height/30
)))+"px;padding-right:"+String(Math.round(7*(height/30)))+"px;justify-
content: center; font-weight: bold; display:flex; align-items:center;
'> <b style='font-size: "+fontsize+"px;'>"+ibValue+"
(A)</b> </div>"
                +"</div>"
}
```

1.3 CÓDIGO PARA GENERAR CLUSTERS, PRUNECLUSTERS Y MARCADORES

Esta función es la encargada tanto de generar los clústeres originales como los PruneClusters además de los marcadores. Primero obtiene los datos de los dispositivos (longitud, latitud y atributos) mediante ajax.

Posteriormente se define la función de generar el icono del nuevo cluster que será diferente en función de la cantidad de marcadores que alberga y su estado. Luego se define la función de generar los clústeres antiguos y finalmente se generan los marcadores añadiéndolos a cada sistema de clúster.

Hay que mencionar que a la hora de generar los marcadores se llama a la función de generar el icono del marcador anteriormente explicado.

```
function generateMarkers(){
            markers = [];
            markars=[];
            return $.ajax({
                type: 'GET',
                url: getContextPath()+url_device_objects.get_device_status,
                data: {groupId: $('#group_selector').val()},
                dataType:'json'
            }).then(function (data) {
                        mapElements = [];
                        for (key in data){
                            var dataJson = eval(data[key]);
                            for(var x in dataJson){
                                //Map info
                                mapElements.push({"tag": dataJson[x].tag,
"sN": dataJson[x].serial_number, "lat": dataJson[x].latitude, "lng":
dataJson[x].longitude,
                                                                    "status":
dataJson[x].status, "isGw": dataJson[x].is_gateway, "nw":
dataJson[x].network});
                            }
                        }
```

```
                        //Prune clusers: cluster con separación de colores
dentro
                        var colors = ['#f7cb1b', '#c0bbb7',
'#009045','#d6270d'],
                        pi2 = Math.PI * 2;
                        L.Icon.MarkerCluster = L.Icon.extend({
                        options: {
                        iconSize: new L.Point(44, 44),
                        className: 'prunecluster leaflet-markercluster-
icon'

                        },
                        createIcon: function() {
                        // based on L.Icon.Canvas from shramov/leaflet-
plugins (BSD licence)
                        var e = document.createElement('canvas');
                        this._setIconStyles(e, 'icon');
                        var s = this.options.iconSize;
                        e.width = s.x;
                        e.height = s.y;
                        this.draw(e.getContext('2d'), s.x, s.y);
                        return e;
                        },
                        draw: function(canvas, width, height) {
                        var start = 0;
                        for (var i = 0, l = colors.length; i < l; ++i) {
                        var size = this.stats[i] / this.population;
                        if (size > 0) {
                        canvas.beginPath();
                        canvas.moveTo(22, 22);
                        canvas.fillStyle = colors[i];
                        var from = start + 0.14,
                        to = start + size * pi2;
                        if (to < from) {
                        from = start;
                        }
                        canvas.arc(22, 22, 22, from, to);
                        start = start + size * pi2;
                        canvas.lineTo(22, 22);
                        canvas.fill();
                        canvas.closePath();
                        }
                        }
                        canvas.beginPath();
                        canvas.fillStyle = 'white';
                        canvas.globalAlpha=1;
                        canvas.arc(22, 22, 15, 0, Math.PI * 2);
                        canvas.fill();
                        canvas.closePath();
                        canvas.fillStyle = '#555';
                        canvas.textAlign = 'center';
                        canvas.textBaseline = 'middle';
                        canvas.font = 'bold 12px sans-serif';
                        canvas.fillText(this.population, 22, 22, 40);
                        }
                        });



                        ClusterPrune = new PruneClusterForLeaflet(20,2);
```

```
                        ClusterPrune.BuildLeafletClusterIcon =
function(cluster) {
                        var e = new L.Icon.MarkerCluster();
                        e.stats = cluster.stats;
                        e.population = cluster.population;


                        return e;
                        };

                        //Clusters originales
                        markerClusters = L.markerClusterGroup({
                            maxClusterRadius: 10 ,
                            spiderfyOnMaxZoom: true,
                            singleMarkerMode: zoomlvl,
                            disableClusteringAtZoom: 7,

                            iconCreateFunction: function(cluster){
                                var childMarkers =
cluster.getAllChildMarkers();
                                var status = "greyCluster";
                                for (var i = 0; i < childMarkers.length;
i++) {
                                    if(childMarkers[i].options.status ===
3){
                                        status = "redCluster" ;
                                        break;
                                    }
                                    else if(childMarkers[i].options.status
=== 2 && status !== "redCluster"){
                                        status = "yellowCluster";
                                    }
                                    else if(childMarkers[i].options.status
=== 0 && status !== "yellowCluster" && status !== "redCluster" ){
                                        status = "greenCluster";
                                    }
                                }
                                statusDef = status + " marker-cluster
marker-cluster-small";
                                return L.divIcon({
                                    html: '<div><span><b>' +
cluster.getChildCount() + '</b></span></div>',
                                    className: statusDef
                                });
                            }
                        });


                        mapElements.map(function(mark){
                            if (map===undefined){
                                markerZoom=3
                            }else{
                                markerZoom= map.getZoom();
                            }

                            //marker = L.marker( [mark.lat, mark.lng],
{title:mark.tag, sn:mark.sN, icon: generateIconToMap(mark), status:
mark.status, tag: mark.tag}).on('click', markerFunctionPopupMap);
```

```
                            var CustomHtmlIcon = new L.DivIcon({html :
generateMarkerHtml(mark.status, mark.tag,markerZoom),iconSize: [91, 30],
iconAnchor: [0,15]});
                                marker = L.marker( [mark.lat, mark.lng],
{title: mark.tag, sn:mark.sN, icon: CustomHtmlIcon, status: mark.status,
tag: mark.tag}).on('click', markerFunctionPopupMap);
                                //var colors = ['#ff00b4', '#ff4b00',
'#00ff4b', '#00b4ff'],
                                //marker.category=mark.status;
                            markers.push(marker);
                            markerClusters.addLayer( marker );


                        markar = new
PruneCluster.Marker(mark.lat,mark.lng,{title: mark.tag, sn:mark.sN,icon:
CustomHtmlIcon,status: mark.status, tag: mark.tag},mark.status);
                        //(lat, lng, data, category, weight, filtered)
                        //markar.data.popup = markerFunctionPopupMap;

                        markars.push(markar);
                        ClusterPrune.RegisterMarker(markar);
                        //heat.push([mark.lat,mark.lng,1]);

                        });


                });
        }
```

1.4 CÓDIGO PARA DIBUJAR EL POLÍGONO EN EL MAPA

Esta función añade la funcionalidad de dibujar al mapa. Para ello crea un botón en la parte superior izquierda que al clickar en el añade un evento 'onClick' al mapa. Cuando salta el evento click, llama a la función de generar el polígono indicando que es el primer turno y la posición donde se ha detectado el evento. Después añade los eventos 'onmousemove' y 'onclick' otra vez y quita el primer evento de click. Estos eventos apuntan a dos funciones diferentes, el 'mousmove' detecta cada movimiento del ratón llamando a la función cuadro que llama a la función el cuadro con turno 2 y posición. Esto hace que cada vez que se mueva el ratón después del primer click se puede ver en el mapa el polígono que se está generando. Finalmente, el evento 'onclick' restante llama a la función clickar2 que llama a la función el cuadro con turno 3 y las coordenadas del punto, quitando tanto el evento de mousemove como el propio de clikcar.

```
L.easyButton('fa-globe', function(btn, map){

                    cuadro = function(e){
                        var latlng =
map.mouseEventToLatLng(e.originalEvent)
                        elcuadro(2,latlng)
                    }
                    clickar2 = function(e){
                        var latlng =
map.mouseEventToLatLng(e.originalEvent)
                        map.off('mousemove', cuadro)
                        map.off('click',clickar2)
```

```
                        elcuadro(3,latlng)
                        puntoCuadro=[]
                    }
                    clickar = function(e) {
                        var latlng =
map.mouseEventToLatLng(e.originalEvent)
                        map.on('mousemove', cuadro);
                        map.on('click',clickar2);
                        map.off('click',clickar);
                        elcuadro(1,latlng);
                        }
                    //map.off('click', doStuff)
                    map.on('click', clickar)


                }).addTo(map);
```

1.5 CÓDIGO PARA EJECUTAR EL POLÍGONO Y EL INICIO LA INTERPOLACIÓN

Esta función es llamada a partir de los eventos que se crean en el paso anterior. Tiene como parámetros el turno o momento en el que se llama y las coordenadas del evento en cuestión. Si el turno es 1 guarda las coordenadas como punto de origen. SI el turno es 2 genera un polígono (rectángulo) desde el punto inicial hasta el segundo punto. Si el turno es 3 hace lo mismo que en el paso 2 pero además genera el polígono y obtiene todos los marcadores dentro del mismo. posteriormente llama a la función callHeatMap añadiendo el resultado al mapa.

```
elcuadro = function(turno,punto){

                    if (turno==1){
                        puntoCuadro.push({lat:decimales(punto.lat),
lng:decimales(punto.lng)})
                    }
                    if (turno==2){
                        puntoCuadro=[puntoCuadro[0]]
                        puntoCuadro.push({lat:
decimales(puntoCuadro[0].lat), lng: decimales(punto.lng)})
                        puntoCuadro.push({lat: decimales(punto.lat),lng:
decimales(punto.lng)})
                        puntoCuadro.push({lat: decimales(punto.lat),
lng:decimales(puntoCuadro[0].lng)})

                        if (pol != []){
                            map.removeLayer(pol)
                            pol = new L.polygon(puntoCuadro)
                            pol.addTo(map)
                        }else{
                            pol = new L.polygon(puntoCuadro)
                            pol.addTo(map)
                        }
                    }
                    if (turno==3){
                    if (map.hasLayer(inter)){

                        layerscontrol.removeLayer(inter)
                        map.removeLayer(inter)
```

```
                    }
                    //coge todos los dispositivos que hay dentro del cuadro
                    var devices=[]
                    markerClusters.eachLayer(function(l) {
                    if( l instanceof L.Marker &&
pol.getBounds().contains(l.getLatLng()) ){
                        devices.push({sn:l.options.sn});
                        }
                    });


                    //para pintar poligonos poner interp en lugar de cors
                    callHeatMap(puntoCuadro,devices).then(function(inter){


                        layerscontrol.addOverlay(inter,"Heatmap")
                        inter.addTo(map)
                        map.removeLayer(pol)
                        //to do: Controlar que haya una leyenda
                        var legend = L.control({position: 'bottomright'});

                        legend.onAdd = function (map) {

                            var div = L.DomUtil.create('div', 'info
legend'),
                            grades = [67.75,65.83, 63.93, 62.03, 60.13,
58.23, 56.33],
                            labels = [];

                        // loop through our density intervals and generate
a label with a colored square for each interval
                            for (var i = 0; i < grades.length; i++) {
                                div.innerHTML +=
                                '<i style="width: 18px; height: 18px;float:
left; margin-right: 8px;opacity: 0.7;background:'+getColor(grades[i] +
1)+'"></i> ' +
                                grades[i] + (grades[i + 1] ? '&ndash;' +
grades[i + 1] + '<br>' : '');
                            }
                            L.DomUtil.get(div).style.background="white";

L.DomUtil.get(div).style.background.opacity=0.7;

                            return div;
                        };


                        map.removeControl(legend);
                        map.addControl(legend)
                        //legend.addTo(map);
                    })

                    }
                }
```

1.6 CÓDIGO QUE LLAMA AL SERVLET PARA LA INTERPOLACIÓN

Esta función es llamada después de generar el polígono y determinar los marcadores que tiene dentro. Esta función se encarga de enviar los datos recibidos al servlet en formato texto y posteriormente de recibir la respuesta y enviarla a la función anterior para introducir el resultado en el mapa. Además los colores que llevará el polígono también se definen aquí.

```
function callHeatMap(hmap,devices){
            return $.ajax({
                type: 'POST',
                url: getContextPath()+"/user_area/servlets/getHeatMap",
                dataType: 'json',
                headers: {'Content-Type': 'application/x-www-form-
urlencoded;charset=utf-8;'},
                data: {devices:JSON.stringify(devices),polygon:
JSON.stringify(hmap)}
            }).then(function(response){
                var parsejs=JSON.parse(response);
                var nivel=[]
                var cors=[]
                function onEachFeature(feature,parsejs){
                    nivel.push(feature.properties.level)
                    cors.push(feature.geometry.coordinates)

                }
                var interp=L.geoJSON(parsejs,{
                    onEachFeature:onEachFeature
                });
                //Para polígonos
                interp=L.geoJSON(parsejs, {

                    style: function(feature) {

                        switch (feature.properties.level) {

                        case nivel[0]: return {color: "#ff0000",weight:
0,fillColor: "#ff0000",smoothFactor:0,fillOpacity: 0.6};
                        //case nivel[0]: return {color:
"#ff0000",smoothFactor: 1};
                        case nivel[1]: return {color:
"#ff5400",smoothFactor: 1};
                        case nivel[2]: return {color:
"#f8d922",smoothFactor: 1};
                        case nivel[3]: return {color:
"#98d922",smoothFactor: 1};
                        case nivel[4]: return {color:
"#00fd01",smoothFactor: 1};
                        case nivel[5]: return {color:
"#2E9AFE",smoothFactor: 1};

                        }
                    }
                });
                console.log(nivel)
                console.log(cors)
                //Para introducir polígonos
                //return interp
                return interp
```

```
        },function(error){
                alert("No data, selecciona 2 marcadores como mínimo");
                map.removeLayer(pol);
        })
    }
```

2. CÓDIGO DE JAVA PARA EL SERVLET

Este es el código java del servlet encargado de recibir los datos del visor y enviarlos al lambda. Este servlet utiliza otros servlets ya implementados para obtener datos como se puede ver en los imports que se hacen al principio.

El servlet recibe los datos del visor, y modifica los textos que han llegado para amoldarlos a las tareas que se van a realizar. Teniendo en cuenta que la información ha llegado como un json pasado a texto tiene que quitar las llaves que lo componen. El polígono lo convierte en una lista de coordenadas que luego serán enviados como texto al AWS lambda.

En cuanto a los dipositivos, determina cuáles de ellos llevan integrado GPS y cuales no. Con esto genera dos listas diferentes y las pasa a formato texto. Luego introduce una fecha de inicio y una fecha fin predefinidas también en formato texto y una vez que tiene todas las variables necesarias las envía al AWS lambda. Después recibe la respuesta, si ha salido bien lo envía de vuelta al visor y si no envía un mensaje de error.

```
package controllersv1.map;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.TimeZone;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.lambda.AWSLambda;
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.model.InvokeRequest;
import com.amazonaws.services.lambda.model.InvokeResult;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
```

```
import connection.DBConnectionPool;

@WebServlet("/user_area/servlets/getHeatMap")
@SuppressWarnings("serial")
public class GetHeatMap  extends HttpServlet{
    public GetHeatMap()
    {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
    {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
    {
        Connection dbConnection = null;
        String userId = request.getUserPrincipal().getName(); //logged user
        String polygon =
request.getParameter("polygon").replace("{\"lat\":",
"[").replace("\"lng\":","").replace("}","]");
        polygon = polygon.substring(1, polygon.length() -1);
        List<String> device_sNs = new ArrayList<String>();
        com.google.gson.JsonArray deviceListAsJsonArray = new
com.google.gson.JsonParser().parse(request.getParameter("devices")).getAsJs
onArray();
        if(deviceListAsJsonArray!=null){
            for(com.google.gson.JsonElement deviceAsJsonElement :
deviceListAsJsonArray)
            {
                com.google.gson.JsonObject deviceAsJsonObject =
deviceAsJsonElement.getAsJsonObject();
                String device_sN =
deviceAsJsonObject.get("sn").getAsString();
                device_sNs.add(device_sN);

            }
        }
        JsonObject objectRequestLambda = new JsonObject();
        System.out.println(polygon);
        if(device_sNs.size()>0) {
            try
            {
                dbConnection = DBConnectionPool.getConnection();
                List<kunak.kunakcloud.dto.Device> devices =
kunak.kunakcloud.dao.Device.readIdentificationInfoBySerialNumber(dbConnecti
on, userId, device_sNs);
                List<kunak.kunakcloud.dto.Device> devicesWithGPS =
getDevicesWithGPS( dbConnection,  devices);
                List<kunak.kunakcloud.dto.Device> devicesWithOutGPS = new
ArrayList<kunak.kunakcloud.dto.Device>();
                Map<Integer,kunak.kunakcloud.dto.Device> mapDevicesWithGPS
= new LinkedHashMap<Integer,kunak.kunakcloud.dto.Device>();
                System.out.println("Devices with GPS");
                String deviceWithGPS = "";
                String deviceWithoutGPS = "";
```

```
                for(kunak.kunakcloud.dto.Device dev : devicesWithGPS) {
                    if(dev.getPartNumber().startsWith("K-A") ||
dev.getPartNumber().startsWith("K-V")) {
                        mapDevicesWithGPS.put(dev.getId(), dev);
                        deviceWithGPS += dev.getId()+",";
                        System.out.println(dev.getId());
                    }
                }
                if(deviceWithGPS.length()>0) {
                    deviceWithGPS = deviceWithGPS.substring(0,
deviceWithGPS.length()-1);
                }
                System.out.println("Devices without GPS");
                for(kunak.kunakcloud.dto.Device dev : devices) {
                    if(!mapDevicesWithGPS.containsKey(dev.getId())) {
                        if(dev.getPartNumber().startsWith("K-A") ||
dev.getPartNumber().startsWith("K-V")) {
                            devicesWithOutGPS.add(dev);
                            deviceWithoutGPS += dev.getId()+",";
                            System.out.println(dev.getId());
                        }

                    }
                }
                if(deviceWithoutGPS.length()>0) {
                    deviceWithoutGPS = deviceWithoutGPS.substring(0,
deviceWithoutGPS.length()-1);
                }

                System.out.println(deviceWithoutGPS);
                System.out.println(deviceWithGPS);
                objectRequestLambda.addProperty("polygon", polygon);
                objectRequestLambda.addProperty("device", deviceWithGPS);
                objectRequestLambda.addProperty("deviceNO",
deviceWithoutGPS);
                objectRequestLambda.addProperty("start_date", "2018-04-05
00:00:00");
                objectRequestLambda.addProperty("end_date","2018-04-05
01:00:00");
                InvokeRequest invokeRequest = new
InvokeRequest().withFunctionName("HeatMap")

.withPayload(objectRequestLambda.toString()).withSdkRequestTimeout(90000).w
ithSdkClientExecutionTimeout(90000);

                BasicAWSCredentials awsCreds = new
BasicAWSCredentials(********,********);

                AWSLambda awsLambda = AWSLambdaClientBuilder.standard()
                        .withRegion(Regions.EU_WEST_1)
                        .withCredentials(new
AWSStaticCredentialsProvider(awsCreds)).build();


                InvokeResult invokeResult = null;
                invokeResult = awsLambda.invoke(invokeRequest);
                ByteBuffer byteBuffer = invokeResult.getPayload();
                String body = new String(byteBuffer.array(), "UTF-8");
                String g = "\\\"";
                System.out.println("body: "+body);
```

```
                if(body.contains("stackTrace")){
                    JsonParser parser = new JsonParser();
                    JsonObject error =
parser.parse(body).getAsJsonObject();
                    JsonObject error2 =
parser.parse(error.get("errorMessage").getAsString()).getAsJsonObject();
                    String message = error2.get("message").getAsString();
                    System.out.println(message);
                    if(message.equals("Error in mutate_impl(.data, dots) :
Evaluation error: subscript out of bounds. ")){
                        response.setHeader("Content-Type", "text/plain");

response.setStatus(error2.get("httpStatus").getAsInt());
                        response.getWriter().print("error");
                    }
                    else{
                        response.setHeader("Content-Type", "text/plain");

response.setStatus(error2.get("httpStatus").getAsInt());
                        response.getWriter().print("error");
                    }

                }
                else if(body.contains("errorMessage") ||
body.contains("message")){
                    response.setHeader("Content-Type", "text/plain");
                    response.setStatus(400);
                    response.getWriter().print("Timeout");
                }
                else if(body.equals("\"None\"")){
                    response.setHeader("Content-Type", "text/plain");
                    response.setStatus(401);
                    response.getWriter().print("No data");
                }
                else{
                    response.setHeader("Content-Type", "text/plain");
                    response.setStatus(200);
                    response.getWriter().print(body);
                }

            }
            catch (Exception e)
            {
                e.printStackTrace();
                response.setStatus(400);
                response.setContentType("application/json;charset=UTF-8");
                response.getWriter().print(new
com.google.gson.Gson().toJson("Paremeters incorrect"));
            }
            finally
            {

org.apache.commons.dbutils.DbUtils.closeQuietly(dbConnection);
            }
        }
        else {
            response.setStatus(401);
            response.setContentType("application/json;charset=UTF-8");
            response.getWriter().print(new
com.google.gson.Gson().toJson("No Devices"));
```

```
        }

    }

    public static List<kunak.kunakcloud.dto.Device>
getDevicesWithGPS(Connection dbConnection,
List<kunak.kunakcloud.dto.Device> devices) throws SQLException {
        PreparedStatement ps = null;
        ResultSet  rs = null;
        String query = "SELECT distinct(device_id) as device_id FROM
kunakcloudv2.element WHERE id=3 and device_id in (";
        Map<Integer, kunak.kunakcloud.dto.Device> mapDevices = new
LinkedHashMap<Integer, kunak.kunakcloud.dto.Device>();
        List<kunak.kunakcloud.dto.Device> devicesWithGPS = new
ArrayList<kunak.kunakcloud.dto.Device>();
        for(kunak.kunakcloud.dto.Device dev : devices) {
            query += dev.getId()+",";
            mapDevices.put(dev.getId(), dev);
        }
        query = query.substring(0, query.length()-1) + ")";
        try
        {
            ps = dbConnection.prepareStatement(query);
            rs = ps.executeQuery();
            while(rs.next())
            {
                Integer device_id = rs.getInt("device_id");
                devicesWithGPS.add(mapDevices.get(device_id));
            }
        }
        catch(SQLException e)
        {
            e.printStackTrace();
            throw e;
        }
        finally
        {
            org.apache.commons.dbutils.DbUtils.closeQuietly(rs);
            org.apache.commons.dbutils.DbUtils.closeQuietly(ps);
        }
        return devicesWithGPS;
    }
}
```

3. CÓDIGO DE PYTHON/R PARA LA INTERPOLACIÓN

En este apartado se muestra el código de Python que se manda a AWS Lambda para que ejecute la interpolación (en el que está embebido el código de R) y devuelva Json con polígonos. Este código está dividido en 3 módulos, Handler, DBConection y HeatMapCalculation. El primero se encarga de recibir los datos y de importar los otros dos módulos. DBConection realiza la conexión a la base de datos para obtener la información de los dispositivos y el HeatMapCalculation ejecuta la interpolación mediante R.

3.1 MÓDULO HANDLER

Importa las librerías de Python que se van a utilizar y posteriormente carga las librerías d R que van a ser necesarias para realizar la interpolación guardadas en 'lib'. Importa rpy2 que es la librería que permite introducir código de R en Python, con la que se puede hacer referencia a objetos que existan en el entorno de R. Esto es necesario para poder operar con los objetos que se vayan generando en R.

Luego introduce los datos recibidos desde el servlet (fechas, dispositivos y polígono) en sus correspondientes variables e intenta ejecutar la función de calculate_heat_map del módulo HeatMapCalculation.

Finalmente espera la respuesta, si hay un error devuelve el error y si todo va correctamente devuelve el Json que ha recibido.

```
# -*- coding: utf-8 -*-
import ctypes
import json
import os

# use python logging module to log to CloudWatch
# http://docs.aws.amazon.com/lambda/latest/dg/python-logging.html
import logging
import DBConnection
import HeatMapCalculation
logging.getLogger().setLevel(logging.DEBUG)

# must load all shared libraries and set the R environment variables before
you can import rpy2

# # load R shared libraries from lib dir

for file_import in os.listdir('lib'):
    if os.path.isfile(os.path.join('lib', file_import)):
        ctypes.cdll.LoadLibrary(os.path.join('lib', file_import))

# set R environment variables

os.environ["R_HOME"] = os.getcwd()
os.environ["R_LIBS"] = os.path.join(os.getcwd(), 'site-library')

# import rpy2

import rpy2
from rpy2 import robjects
from rpy2.robjects import r
from datetime import datetime


def lambda_handler(event, context):
    device = event['device']
    deviceNO = event['deviceNO']
    polygon = event['polygon']
    start_date = event['start_date']
    end_date = event['end_date']
    print(start_date)
```

```
    try:
        jsonHeatMap = HeatMapCalculation.calculate_heat_map(r, robjects,
device, deviceNO, start_date, end_date, polygon)
        return jsonHeatMap
    except Exception, e:
        DBConnection.disconnected_to_database(r)
        logging.error('Error HeatMap calculation: {0}'.format(e.message))
        error = dict()
        error['errorType'] = 'HeatMapException'
        error['httpStatus'] = 400
        error['request_id'] = context.aws_request_id
        error['message'] = "HeatMap error"
        error['message_err'] = e.message.replace('\n', ' ')
        raise Exception(json.dumps(error))
    return error
```

3.2 MÓDULO HEATMAPCALCULATION

Este módulo es el encargado de recibir los datos que le envía el Handler y define funciones en R que se utilizarán posteriormente. La función principal de R es interpolate, que será el encargado de partiendo de las coordenadas en formato texto, generar un objeto de SpatialPolygonDataFrame en R, además de recibir los datos de los dispositivos y realizar la interpolación.

Para poder ejecutar la función interpolate la primera que se ejecuta es calculate_heat_map que es llamada desde el módulo handler. Esta función se encarga de llamar al módulo DBConnection y ejecutar las funciones init_libraries y init_functions que definen las funciones de conexiones a la base de datos. Posteriormente llama a la función _get_reads definida más abajo en el módulo HeatMapCalculation que es el encargado de coger las lecturas que están en la base de datos para los dispositivos que han llegado, utilizando las funciones generadas en R a partir de DBConnection.

Posteriormente se llama a la función _calculate_heat_map que realiza la interpolación guardando el resultado en la variable jsonHeatMap y finalmente cierra la conexión con la base de datos devolviendo como resultado jsonHeatMap.

```
import logging
import DBConnection
import json

def __init_functions(r):
    r("interpolate<- function(deviceReads.ag,polygon){\n"
      "fich.trabajo<-deviceReads.ag[,c(\"Lat\",\"Lng\",\"PM10\")]\n"
      "coordinates(fich.trabajo)<-~Lng+Lat\n"
      "proj4string(fich.trabajo)=CRS(\"+proj=longlat\")\n"
      "textjson<-gsub(\",[:punct:]\", \"\", polygon)\n"
      "textjson<-gsub(\"[:punct:]\",\"\",textjson)\n"
      "textjson<-strsplit(textjson, split=\"]\")\n"
      "jlist<-matrix(0, ncol = 2, nrow = 0)\n"
      "jlist<-data.frame(jlist)\n"
      "names(jlist)<-c(\"Lng\",\"Lat\")\n"
      "for (i in textjson[[1]]){\n"
      "spliter<-strsplit(i[1],split=\",\")\n"
      "fila<-
data.frame(as.numeric(spliter[[1]][2]),as.numeric(spliter[[1]][1]))\n"
      "names(fila)<-c(\"Lng\",\"Lat\")\n"
```

```
        "jlist<-rbind(jlist,fila)}\n"
        "data<-data.frame(id=1)\n"
        "jlistdf<-Polygon(jlist)\n"
        "poligonos<-Polygons(list(jlistdf),1)\n"
        "spoligons<-SpatialPolygons(list(poligonos))\n"
        "pv.SPDF<-SpatialPolygonsDataFrame(spoligons,data=data, match.ID =
FALSE)\n"
        "proj4string(pv.SPDF)<-CRS(\"+proj=longlat\")\n"
        "malla.grd = SpatialPixels(SpatialPoints(makegrid(pv.SPDF, n =
50000)),proj4string = proj4string(pv.SPDF))\n"
        "grid0<-malla.grd\n"
        "grid0<-raster(grid0)\n"
        "interpolation = lgm( PM10~ 1,fich.trabajo,grid0,aniso=F)\n"
        "predict.raster = raster::mask(interpolation$predict[[\"predict\"]],
pv.SPDF)\n"
        "haz.cero.na=function(x){\n"

"ifelse(x<(mini+i),0,ifelse(x<(mini+i*2),1,ifelse(x<(mini+i*3),2,ifelse(x<(
mini+i*4),3,ifelse(x<(mini+i*5),4,5)))))}\n"
        "maxi<-max(values(predict.raster$predict),na.rm = TRUE)\n"
        "mini<-min(values(predict.raster$predict),na.rm=TRUE)\n"
        "dif<-maxi-mini\n"
        "i<-dif/6\n"
        "predict.raster$pol<-sapply(predict.raster$predict,haz.cero.na)\n"
        "x<-predict.raster$pol\n"
        "pol0<-rasterToPolygons(x, fun=function(x){x==0}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "pol1<-rasterToPolygons(x, fun=function(x){x==1}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "pol2<-rasterToPolygons(x, fun=function(x){x==2}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "pol3<-rasterToPolygons(x, fun=function(x){x==3}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "pol4<-rasterToPolygons(x, fun=function(x){x==4}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "pol5<-rasterToPolygons(x, fun=function(x){x==5}, n=16, na.rm=TRUE,
digits=12, dissolve=TRUE)\n"
        "polis<-rbind(pol0,pol1,pol2,pol3,pol4,pol5)\n"
        "exportJson <- as.geojson(polis)\n"
        "exportJson<- as.character(exportJson)\n"
        "exportJson}\n")

def calculate_heat_map(r, robjects, device, deviceNO, start_date, end_date,
polygon):
    __init_functions(r)
    DBConnection.init_libraries(r)
    DBConnection.init_functions(r)
    __get_reads(r, robjects, device, deviceNO, start_date, end_date)
    jsonHeatMap = __calculate_heat_map(r, robjects,polygon)
    DBConnection.disconnected_to_database(r)
    return jsonHeatMap

def __get_reads(r, robjects, device, deviceNO, start_date, end_date):
    logging.info('Init get reads from {0}'.format(device))
    r("dataRaw <-
executeQuery(toString(\""+start_date+"\"),toString(\""+end_date+"\"),\""+de
vice+"\",\""+deviceNO+"\")")
    if robjects.r['dataRaw'] != robjects.rinterface.NULL:
        logging.info("Finished get reads, size:
{0}".format(str(robjects.r['dataRaw'].nrow)))
```

```
    else:
        logging.info("Finished get reads, size: 0")

def __calculate_heat_map(r, robjects,polygon):
    logging.info('Start calculate HeatMap')
    if robjects.r('dataRaw') != robjects.rinterface.NULL:
        r('jsonHeatMap <- interpolate(dataRaw,"'+polygon+'")')
    logging.info('End  calculate HeatMap')
    if robjects.r('dataRaw')!=robjects.rinterface.NULL:
        r('rm(dataRaw)')
        jsonHeatMap =
json.loads(str(robjects.r['jsonHeatMap']).replace("[1] ",""))
        return jsonHeatMap
    return "None"
```

3.3 MÓDULO DBCONECTION

Este módulo es el encargado de definir las funciones de R que generan las conexiones y consultas necesarias para poder obtener las lecturas de los dispositivos cuyas ID han llegado a Lambda desde el servlet. Los datos de usuarios y contraseñas no se muestran en el código por motivos de seguridad.

Se crean funciones diferentes en R en función de si los dispositivos tenían GPS incorporado o no. Ese dato se sabe de antemano pues es el servlet el encargado de separarlos en dos listas.

Si los dispositivos tienen GPS llama a la función getReadsOfDevicesWithGPS que llama a su vez a getQueryReads. getQueryReads es el encargado de realizar la consulta a la tabla read de la base de datos obteniendo las lecturas para los id de los elementos que se han seleccionado. Como los dispositivos de este grupo tienen GPS se incluyen las ID de los elementos correspondientes a longitud y latitud que existen en la tabla además del contaminante que en este caso es PM10. Posteriormente trata los datos recibidos generando un objeto de R DataFrame con los mismos.

Para los dispositivos sin GPS, se llama a la función getReadOfDevicesNoGPS que nuevamente llama a getQueryReads obteniendo las lecturas de los dispositivos. La diferencia es que en este caso no se añaden las ID de los elementos correspondientes a las coordenadas puesto que no existen. En su lugar se llama a una segunda función llamada getQueryCoords que hace una consulta en la tabla device de la base de datos donde se encuentran los datos de longitud y latitud de estos dispositivos. Una vez que se tienen tanto las lecturas como las coordenadas se crea un DataFrame igual que en el caso anterior.

Una vez que los dos DataFrames están definidos, se juntan en uno solo que tiene todas las lecturas para cada dispositivo desde la fecha inicio hasta la fecha fin. Como se necesita un único valor por dispositivo para poder realizar la interpolación, se hace una media de los valores registrados en esa franja horaria y se asigna ese valor a cada dispositivo con la función aggregate de R.

Finalmente se devuelve esta tabla de datos. Tanto la funcion de connect_to_database, como disconnected_to_database, son llamadas desde el módulo de HeatMapCalculation y son encargadas de crear la conexión a la base de datos y de realizar la desconexión de la misma.

```
import logging

userDbAmazon = ****
hostDbAmazon = ****
pwdDbAmazon = ****
dbnameDbAmazon = ****


def init_libraries(r):
    r('library(DBI)')
    r('library(RMySQL)')
    r('library(Matrix)')
    r('library(raster)')
    r('library(sp)')
    r('library(geostatsp)')
    r('library(rgeos)')
    r('library(geojson)')

def init_functions(r):

    r("getQueryReads <- function(name, elementId, date1, date2, devices) {
\n"
        "str_device=\"\"\n"
        "for(id in devices){\n"
        "str_device = paste(str_device,\"\",toString(id),\",\")}\n"
        "str_device<- substr(str_device, 0, nchar(str_device)-1)\n"
        "query <- paste(\"SELECT `read`.`date`, `read`.`device_id`,
`read`.value as \",name,\" FROM `read` WHERE `read`.device_id in
(\",str_device,\")  AND `read`.element_id=\",toString(elementId),\" AND
`read`.`date` between \",date1,\" AND \",date2,\" \")\n"
        "query}\n")
    r("getQueryCoords <- function(devices){\n"
        "str_device =\"\"\n"
        "for (id in devices){\n"
        "str_device = paste(str_device,\"\",toString(id),\",\")}\n"
        "str_device<- substr(str_device, 0, nchar(str_device)-1)\n"
        "query <- paste(\"SELECT id as device_id,latitude as Lat, longitude
as Lng FROM kunakcloudv2.device WHERE id in(\",str_device,\")\")\n"
        "query}\n")
    r("getReadOfDevicesNoGPS<-function(con,dateStart,dateEnd, devices){\n"
        "names_sensors <- c(\"PM10\")\n"
        "ids_sensors <- c(49)\n"
        "sensors_name_Ids <- setNames(as.list(ids_sensors),
names_sensors)\n"
        "mydata <- NULL\n"
        "for(name in names(sensors_name_Ids)){ \n"
        "if(is.null(mydata)){\n"
        "res<-dbSendQuery(con, getQueryReads(name
,sensors_name_Ids[[name]],dateStart,dateEnd,devices))\n"
        "dataframe1<- fetch(res, n = -1)\n"
        "dbClearResult(res)\n"
        "mydata <- rbind(mydata,dataframe1)}\n"
        "else{\n"
        "res<-dbSendQuery(con, getQueryReads(name
,sensors_name_Ids[[name]],dateStart,dateEnd,devices))\n"
        "dataframe1<- fetch(res, n = -1)\n"
        "dbClearResult(res)\n"
        "mydata <-
merge(mydata,dataframe1,by=c(\"device_id\",\"date\"))}\n"
        "rm(res, dataframe1)}\n"
```

```
        "cor<-dbSendQuery(con,getQueryCoords(devices))\n"
        "dataframe2<-fetch(cor,n=-1)\n"
        "dbClearResult(cor)\n"
        "mydata <- merge(mydata, dataframe2, by=\"device_id\" )\n"
        "mydata$date <- as.POSIXct(strptime(mydata$date, format = \"%Y-%m-
%d %H:%M:%S\", tz = \"GMT\"))\n"
        "mydata}\n")
    r("getReadOfDevicesWithGPS <- function(con,dateStart,dateEnd,
devices){\n"
        "names_sensors <- c(\"Lat\", \"Lng\", \"PM10\")\n"
        "ids_sensors <- c(3, 4, 49)\n"
        "sensors_name_Ids <- setNames(as.list(ids_sensors),
names_sensors)\n"
        "mydata <- NULL\n"
        "for(name in names(sensors_name_Ids)){ \n"
        "if(is.null(mydata)){\n"
        "res<-dbSendQuery(con, getQueryReads(name
,sensors_name_Ids[[name]],dateStart,dateEnd,devices))\n"
        "dataframe1<- fetch(res, n = -1)\n"
        "dbClearResult(res)\n"
        "mydata <- rbind(mydata,dataframe1)}\n"
        "else{\n"
        "res<-dbSendQuery(con, getQueryReads(name
,sensors_name_Ids[[name]],dateStart,dateEnd,devices))\n"
        "dataframe1<- fetch(res, n = -1)\n"
        "dbClearResult(res)\n"
        "mydata <-
merge(mydata,dataframe1,by=c(\"device_id\",\"date\"))}\n"
        "rm(res, dataframe1)}\n"
        "mydata$date <- as.POSIXct(strptime(mydata$date, format = \"%Y-%m-
%d %H:%M:%S\", tz = \"GMT\"))\n"
        "mydata}\n")
    r("executeQuery<- function(dateStart,dateEnd,devices,devicesNo,con){\n"

        "dateStart = paste(\"'\",dateStart,\"\");\n"
        "dateStart = paste(dateStart,\"'\",\"\");\n"
        "dateEnd = paste(\"'\",dateEnd,\"\");\n"
        "dateEnd = paste(dateEnd,\"'\",\"\");\n"
        "if (devices != \"\"){\n"
        "l<-c(strsplit(devices,\",\"))\n"
        "devices=c()\n"
        "for (i in 1:length(l[[1]])){\n"
        "devices[i]=as.numeric(l[[1]][i])}\n"
        "driverMySql <- dbDriver(\"MySQL\")\n"
        "deviceReads1 <- getReadOfDevicesWithGPS(con,dateStart,dateEnd,
devices);\n"
        "deviceReads <- deviceReads1;\n"
        "rm(l);}\n"
        "if (devicesNo != \"\"){\n"
        "l<-c(strsplit(devicesNo,\",\"))\n"
        "devicesNo=c()\n"
        "for (i in 1:length(l[[1]])){\n"
        "devicesNo[i]=as.numeric(l[[1]][i])}\n"
        "driverMySql <- dbDriver(\"MySQL\")\n"
        "deviceReads2<-
getReadOfDevicesNoGPS(con,dateStart,dateEnd,devicesNo);\n"
        "deviceReads <- deviceReads2;\n"
        "rm(l);}\n"
        "if(devices!=\"\"&&devicesNo!=\"\"){\n"
        "deviceReads<-rbind(deviceReads1,deviceReads2)}\n"
```

```
        "if(exists(\"deviceReads\")&&nrow(deviceReads)!=0){\n"
        "deviceReads.ag<-
aggregate(cbind(PM10,Lng,Lat)~device_id,mean,data=deviceReads)}\n"
        "else{\n"
        "deviceReads.ag <- NULL}\n"
        "if (nrow(deviceReads.ag)<=1){\n"
        "deviceReads.ag<-NULL}\n"
        "deviceReads.ag}\n")

def connect_to_database(r):
    r("driverMySql <- dbDriver(\"MySQL\")")
    r("con <-
dbConnect(driverMySql,user='"+userDbAmazon+"',password='"+pwdDbAmazon+"',ho
st='"+hostDbAmazon+"',dbname='"+ dbnameDbAmazon + "')")
    logging.info("Init connection DB")

def disconnected_to_database(r):
    r("if (exists(\"con\")){\n"
    "dbDisconnect(con)\n"
    "rm(con, driverMySql, getQueryReads, executeQuery)\n}\n")
    logging.info("Close connection DB")
```